SZCZEPANOWSKI

CPC 6128 PARA PRINCIPIANTES

UN LIBRO DATA BECKER

EDITADO POR FERRE MORET, S.A.

SZCZEPANOWSKI

CPC 6128 PARA PRINCIPIANTES

UN LIBRO DATA BECKER
EDITADO POR FERRE MORET, S.A.

Este libro ha sido traducido por Dña Petra Scheffler.

Imprime: APSSA, ROCA UMBERT, 26 L'HOSPITALET DE LL. (Barcelona)

ISBN 84-86437-54-7

Depósito legal — B-21073/86

Advertencia importante

Los circuitos, procedimientos y programas reproducidos en este libro son divulgados sin tener en cuenta el estado de las patentes. Están destinados exclusivamente al uso amateur o docente, y no pueden ser utilizados para fines comerciales. Todos los circuitos, datos técnicos y programas de este libro, han sido elaborados o recopilados con el mayor cuidado por el autor y reproducidos utilizando medidas de control eficaces. No obstante, es posible que exista algún error. FERRE MORET, S.A. se ve por tanto obligada a advertirles, que no puede asumir ninguna garantía, ni responsabilidad jurídica, ni cualquier otra responsabilidad sobre las consecuencias atribuibles a datos erróneos. El autor les agradecerá en todo momento la comunicación de posibles fallos.

CAPITULO 1:

EL TECLADO

CAPITULO 2:

EL PRIMER COMANDO

CAPITULO 3:

EL PRIMER PROGRAMA

CAPITULO 4:

AYUDAS PARA LA PROGRAMACION

CAPITULO 5:

INTRODUCCION AL BASIC

CAPITULO 8:

LA UNIDAD DE DISCO

CAPITULO 9:

MAS COMANDOS

APENDICE

Capítulo 1:

EL TECLADO



Este capítulo le enseñará el manejo de las 74 teclas de su CPC6128. Con ayuda de estas teclas se le abrirán las puertas a las posibilidades ocultas de su ordenador doméstico. Tanto si se trata de la entrada de textos, como de la confección de imágenes (gráficos), del envío de instrucciones o del cálculo de problemas matemáticos, todo esto ya no será nada nuevo para usted después de leer este capítulo. Una vez domine el uso de todas las teclas, el CPC6128 le será de gran ayuda a la hora de solucionar problemas de toda índole.

El dominio del teclado no sólo sirve a los genios, que aspiran a conocer cada Bit del ordenador por su nombre, sino también a aquellos usuarios, que simplemente deseen "alimentarlo" con los programa elaborados que se venden en el mercado. Hay que evitar, que la persona que se cuenta entre estas últimas, no empiece a consultar deseperadamente el manual cuando se encuentra con el mensaje de un programa estándar, como por ejemplo, "PULSE LA TECLA RETURN PARA IMPRIMIR EL CONTENIDO DE LA PANTALLA".

En resumen, cualquier persona que utilice su ordenador, aunque sea de forma esporádica, debería estar familiarizada con el teclado. El padre de un hijo aficionado al ordenador debería saber al menos cómo cargar su mejor programa de juegos, cuando el hijo ha salido de casa.

Pero no se preocupe, no es necesario que se inscriba en un cursillo de máquina de escribir. La mayoría de los programadores aficionados, por mucha expertos que sean, trabajan con el sistema de "dos dedos". La persona que se haya familiarizado un poco con el teclado, se sorprenderá de la rapidez con que va recorriendo el teclado.

En este capítulo dedicaremos una especial atención a las blancas teclas auxiliares. Dichas teclas tienen gran importancia, ya que sirven p.ej. para procesar textos, interrumpir programas y transmitir sentencias al ordenador.

GENERALIDADES SOBRE EL TECLADO

A primera vista, el teclado del CPC6128 da la impresión de ser un teclado normal y corriente de una máquina de escribir. Pero, si se fija bien, observará algunas diferencias:

El teclado no presenta acentos ni letras especiales (ñ,¿,¡,etc), puesto que estos caracteres no existen en el teclado americano.

Existen teclas adicionales (teclas auxiliares), cuya función se explicará más adelante.

Es aconsejable no hacer experimentos con el teclado, antes de familiarizarse mínimamente con su función. Aunque es verdad, que un manejo erróneo no provoca que su ordenador expida nubes de humo, los sorprendentes fracasos pueden dejarle asombrado. Por lo tanto, siga con paciencia la introducción práctica de las siguientes páginas.

ALLA VA

```
Amstrad 128K Microcomputer    (v3)

 1985 Amstrad Consumer Electronics plc
         and Locomotive Software Ltd.
BASIC 1.1

Ready
_
```

Ya es hora de dar vida a la técnica dormida de su CPC6128. Para conectarlo encontrará dos interruptores, uno en el monitor y otro en el ordenador. Hay que conectar los dos. Después de cada conexión aparece un mensaje, que nos indica, que el ordenador está preparado para seguir sus instrucciones. La primera línea del mensaje indica, que está usted trabajando con el ordenador AMSTRAD, que dispone de una memoria principal de 128 KBytes (unos 131070 caracteres). De estas 131070 posiciones de memoria, unas 40000 están reservadas para los programas BASIC. El resto de los primeros 64 KBytes de la memoria es utilizado por el ordenador para el sistema operativo y el lenguaje de programación BASIC. El segundo grupo de 64 KBytes puede ser utilizado como memoria de datos.

La siguiente línea indica la procedencia del ordenador, la empresa AMSTRAD de Gran Bretaña. De la empresa LOCOMOTIVE SOFTWARE procede el BASIC integrado, que, como verá más adelante, es muy potente. Esta versión del BASIC está designada con el número 1.1. Cuando se efectúen modificaciones generales o mejoras de dicha versión, también cambiará este número. El mensaje 'READY' de la última línea indica, que el sistema operativo está preparado para recibir

instrucciones. En general, el mensaje 'READY' señala la disponibilidad del ordenador.

Hablemos ahora del pequeño cuadrado debajo de 'READY'. Se trata de una ayuda de orientación sobre la pantalla. Supongamos, que usted desee escribir algo en la pantalla. Necesitará saber exactamente, en qué posición aparecerá el carácter a entrar. Esta marca, que en el lenguaje especializado se llama CURSOR, le ayudará pues a orientarse en la pantalla.

MANDO DE TRES DEDOS



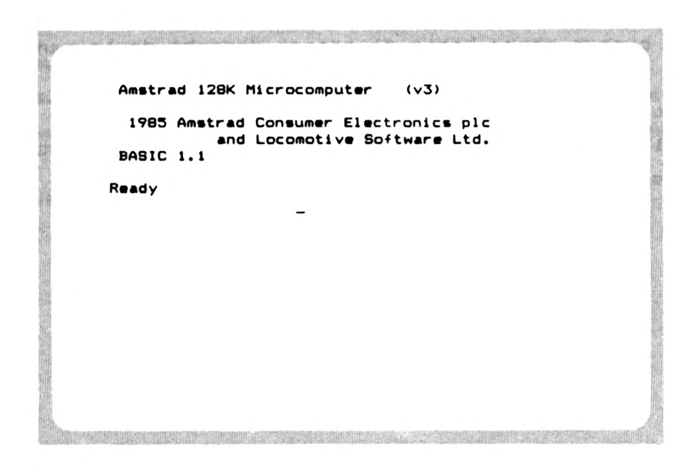
En muchas situaciones es necesario poner el ordenador en su estado original. Muchos ordenadores requieren ser apagados y conectados de nuevo para conseguirlo. Sin embargo, este proceso supone una carga excesiva para su electrónica de alta calidad. Para evitarlo, se puede provocar el estado original del ordenador con ayuda de algunas teclas. Se trata de pulsar las tres teclas SHIFT-CTRL-ESC. Pero debe tener la precaución de que ESC sea la última en ser pulsada, mientras se mantengan las teclas SHIFT y CTRL. Pruebe ahora el mando de tres dedos. Verá, que aparece la misma imagen que tras la conexión del ordenador.

Es conveniente realizar el restablecimiento del ordenador, también llamado RESET o arranque en frío, con precaución, puesto que se borran todos los programas BASIC con el proceso.

LAS TECLAS DEL CURSOR



Estas teclas están situadas debajo del bloque de cifras en la parte derecha del teclado. Las flechas marcadas en ellas indican la dirección que debe seguir el movimiento del cursor. Ahora pulse 20 veces la tecla DERECHA. El cursor se desplazará 20 posiciones hacia la derecha, y volverá a esperar pacientemente en el centro de la pantalla.



¿Pero qué ocurre, si el cursor traspasa el margen derecho de la línea? Pruébelo usted mismo, pulsando de nuevo 20 veces la tecla DERECHA. Después de la pulsación 20 de dicha tecla, el cursor aparecerá en la primera columna de la línea siguiente.

```
Amstrad 128K Microcomputer    (v3)

 1985 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.
BASIC 1.1

Ready

_
```

Es bastante engorroso golpear 20 veces la tecla del cursor para conseguir, que algo se desplace al centro de la pantalla. Se puede simplificar el proceso, manteniendo la tecla pulsada, lo cual provoca el movimiento continuo del cursor. Lo mismo vale para las demás teclas. Pruébelo usted mismo, manteniendo la tecla CURSOR-DERECHA pulsada. Observe, como el cursor corre hacia el margen derecho de la pantalla.

Para mover al cursor ahora en la dirección opuesta, utilice la tecla CURSOR-IZQUIERDA. Si prueba ahora dicha tecla, el cursor se moverá en sentido opuesto.

```
Amstrad 128K Microcomputer   (v3)

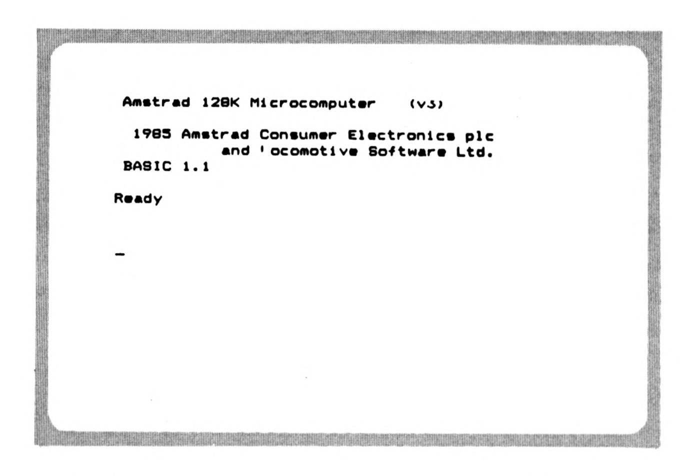
 1985 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.
BASIC 1.1

Ready
```

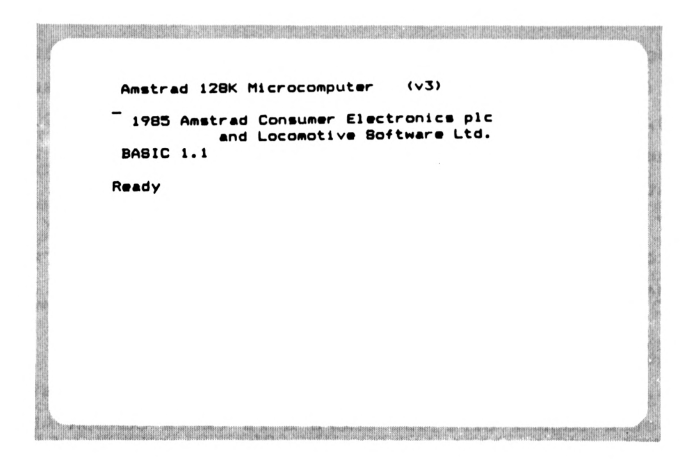
Naturalmente, el cursor también se mueve automáticamente hacia la izquierda, sin nencesidad de golpear repetidamente la tecla correspondiente. Simplemente mantégala pulsada.

¿Cómo podemos conducir el cursor ahora a su posición de salida (debajo de 'READY')? Una forma de conseguirlo consiste, por ejemplo, en pulsar la tecla CURSOR-IZQUIERDA, hasta que el cursor se desplace línea por línea hacia arriba. Pero este procedimiento no es ideal. Las teclas de cursor ARRIBA Y ABAJO permiten mover el cursor hacia arriba y hacia abajo con la misma rapidez, que se movía hacia derecha e izquierda con las teclas descritas anteriormente.

Ahora mueva el cursor hacia el margen izquierdo de la pantalla. El cursor espera con paciencia, hasta que usted lo lleve a otro lugar. Diríjase ahora a la tecla CURSOR-ABAJO y púlsela tres veces. Habrá notado, que el cursor ha saltado tres líneas en sentido descendente.



Ahora intente llevar el cursor hacia la tercera línea de la pantalla. Utilice la tecla CURSOR-ARRIBA.



De nuevo, el cursor avanza automáticamente línea por línea, si mantiene dicha tecla pulsada.

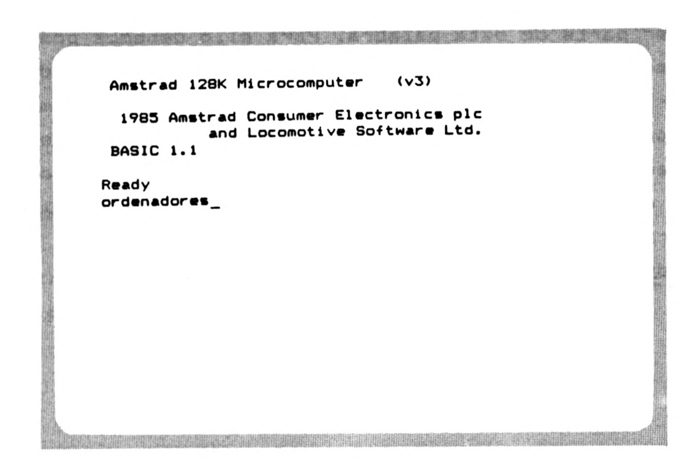
Para adquirir mayor habilidad en el movimiento del cursor, intente ahora dibujar un cuadrado invisible con el mismo. Es decir, mueva el cursor a derecha / abajo / izquierda / arriba, o al revés, izquierda / abajo / derecha / arriba. Es conveniente adquirir el dominio de las teclas de cursor, puesto que más adelante le servirá para realizar p.ej. correcciones con suma rapidez. Pero ya sabemos, que la

práctica hace al maestro...

EDITAR CON LAS TECLAS DE CURSOR

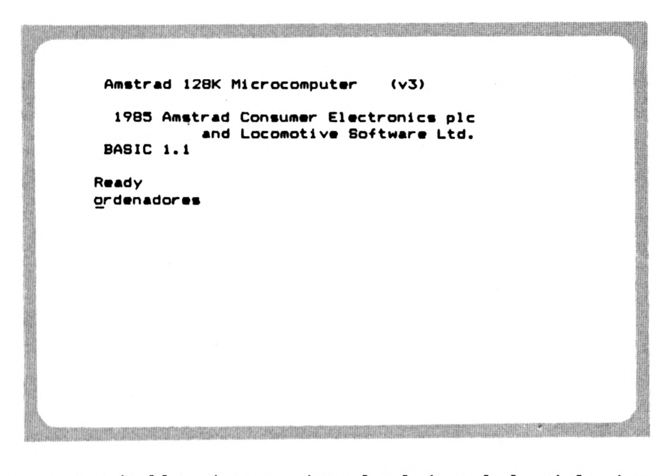
Aquí aparece un tecnicismo del proceso de datos, cuyo significado será una incógnita para algún lector, sobre todo para los principiantes. Editar significa tratar textos, por ejemplo confeccionarlos y modificarlos. El hecho de escribir algunas palabras en la pantalla ya se denomina editar.

Gran parte de los ordenadores le permiten mover el cursor en la pantalla como quiera, entrar texto y continuar el movimiento. Sin embargo, el CPC6128 tiene una característica especial. Le permite mover el cursor de cualquier modo, hasta el momento de pulsar cualquier carácter del teclado. A partir de ese momento sólo podrá mover el cursor dentro de los caracteres entrados. Esto es un poco de teoría gris, que vamos a convertir ahora en práctica. En primer lugar, ponga el ordenador en su estado original de conexión (SHIFT-CTRL-ESC). Ahora escriba la palabra "ordenadores" en la pantalla, aunque tarde un poco en encontrar las letras:

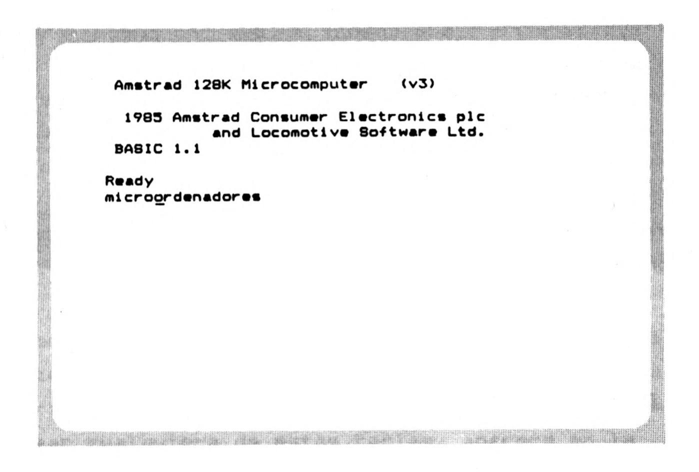


El cursor permanece ahora detrás de la última letra de dicha palabra, negándose decididamente a tomar cualquier dirección diferente a la izquierda, donde se hallan las letras introducidas. Inténtelo, aunque no tendrá éxito. A partir de este momento, el cursor únicamente sirve para tratar el texto entrado. Para ello le prohíbe e impide abandonar el texto. Ahora mueva el cursor al extremo izquierdo, manteniendo la tecla IZQUIERDA. Cuando llegue al margen izquierdo de la pantalla, el cursor se detiene como si hubiera tropezado con una pared. Se producirá una reacción idéntica, si intenta conducirlo hacia la derecha. También parece haber una pared detrás de la 'última letra. ¿Qué razón debe tener esta limitación del movimiento del cursor? Antes de explicar las razones, usted debe entender claramente el fin que persigue al escribir algo en la pantalla. Existen varias razones, tales como p.ej. la confección de programas. Para escribir un programa, lo cual exlicaremos más adelante, entramos varias líneas de programa. Cada línea se transmite al ordenador mediante una tecla determinada (RETURN). No tiene sentido continuar una línea empezada del programa en cualquier otro lugar de la pantalla. Sólo la transferencia de la línea al ordenador volverá a liberar al cursor.

Volvamos a nuestro "ordenadores". Podemos insertar letras en cualquier parte de esta palabra. Para ello debemos posicionar el cursor sobre la letra, delante de la cual deseamos escribir otra letra o carácter. Este modo, llamado de inserción, siempre está activado. Existen ordenadores, que requieren la activación previa de dicho modo. Cambiemos ahora el término "ordenadores", convirtiéndolo en "microordenadores". Para ello lleve el cursor al margen izquierdo de la pantalla.



El cursor se halla ahora sobre la letra 'o', delante de la cual debe insertarse "micro". Vaya pulsando ahora las letras 'm', 'i', 'c', 'r', 'o'.



Cada letra entrada desplaza el texto anterior hacia la derecha, pero el cursor se mantiene sobre la "o".

```
Amstrad 128K Microcomputer    (v3)

 1985 Amstrad Consumer Electronics plc
         and Locomotive Software Ltd.
BASIC 1.1

Ready
microordenadores_
```

Si ahora movemos el cursor al extremo derecho, podemos continuar la entrada de texto. Ya mencionamos antes, que una entrada se finaliza con la tecla 'RETURN'. Sin embargo, si pulsa dicha tecla en nuestro ejemplo, el CPC protesta. Con el mensaje "Syntax error" nos comunica, que "no puedo entender, lo que acabas de decirme". Y es lógico, puesto que el ordenador CPC6128 sólo habla el lenguaje BASIC, que usted deberá aprender antes de poder comunicarse con él. Pero todavía faltan algunas páginas para llegar a ese punto.

Ahora ya conoce el modo de inserción. Si practica un poco más, pronto dominará la "inserción" perfectamente.

BORRADO DE LA PANTALLA

Si usted determina en algún momento, que todo lo que está escrito en el pantalla no son más que galimatías inútiles, bórrelo todo sencillamente. ¡¡ALTO!! No mediante el paño limpiacristales, que puede servirle para eliminar el polvo de la pantalla, pero no los datos tecleados y, por tanto, almacenados en el ordenador. Al fin y al cabo, para un ordenador capaz de procesar aproximadamente 1 millón de instrucciones por segundo, el hecho de borrar los 1000 caracteres de la pantalla con una sentencia no debe representarle ninguna dificultad. Muchos ordenadores disponen

de una tecla para borrar la pantalla. Pero el CPC6128 no es del todo tan confortable, puesto que exige cuatro pulsaciones de tecla. El BASIC del CPC6128 dispone de un comando, que ordena "borrar toda la pantalla" al ordenador. Dicho comando es 'CLS', que viene del inglés (Clear Screen), al igual que todos los comandos. Tal como ya hemos explicado, usted debe cerrar los comandos con la tecla 'RETURN'. Por lo tanto, entre 'CLS' seguido de 'RETURN'. Su CPC le obedece inmediatamente borrando la pantalla. El mensaje 'READY' indica de nuevo, que el ordenador está esperando más instrucciones. Es conveniente memorizar este comando, ya que será utilizado repetidas veces a lo largo del libro.

LAS TECLAS DE LETRAS



Tal como hemos mencionado anteriormente, la disposición de estas teclas se asemeja, salvo ligeros cambios, a una máquina de escribir. Pongamos nuestra atención inicial en las letras. Al pulsar cualquiera de estas teclas, aparece en la pantalla su correspondiente letra minúscula en la posición del cursor. Antes de utilizarlas, borre la pantalla con el comando 'CLS'.

Ahora ya disponemos de una pantalla "limpia" para empezar a editar. Por favor, no entre ahora cantidades inmesurables de texto con la esperanza, de que su ordenador lo acepte todo en su memoria, dispuesto a ser reclamado. La cosa no es tan sencilla. Todo lo que usted entre, simplemente será almacenado en la fugaz memoria de pantalla. En la introducción al BASIC le enseñaremos la forma de memorizar textos entrados.

Ahora escriba todas las letras en orden alfabético en la pantalla. Se dará cuenta, de la dificultad que le depara al principio la localización de una u otra letra, si no tiene la práctica de una secretaria. En la pantalla se hallan ahora todas las letras minúsculas disponibles.

```
Amstrad 128K Microcomputer    (v3)

 1985 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.
BASIC 1.1

Ready
abcdefghijklmnopqrstuvwxyz_
```

Vuelva a limpiar la pantalla y escriba nuevamente todas las letras. Pero esta vez no las escriba por orden alfabético, sino por orden de colocación en el teclado. Eso significa empezar por la línea superior (de Q a P), después la línea central (de A a L), y finalmente la inferior (de Z a M). Seguramente le será más fácil seguir este orden.

```
Amstrad 128K Microcomputer    (v3)

 1985 Amstrad Consumer Electronics plc
         and Locomotive Software Ltd.
BASIC 1.1

Ready
qwertyuiopasdfghjklzxcvbnm_
```

LETRA MAYUSCULA/MINUSCULA



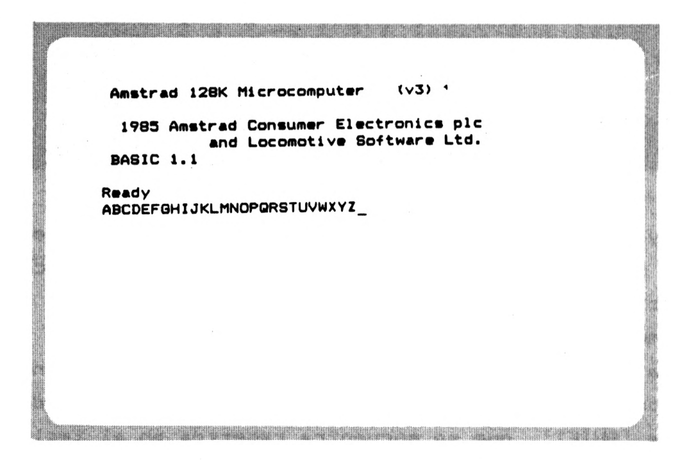
Usted ya habrá notado, que cualquier letra que escriba, aparece en la pantalla como letra minúscula. Con frecuencia, y especialmente en el tratamiento de textos, también precisamos letras mayúsculas. La máquina de escribir cuenta con una tecla adicional, que debe pulsarse simultáneamente para imprimir las mayúsculas. El CPC también dispone de ella. La tecla 'SHIFT', que se encuentra repetida en el teclado, se encarga de las mayúsculas. Por lo tanto, mantenga dicha tecla si desea escribir letras mayúsculas. Ahora escriba el nombre "Carlos Ruiz" en la pantalla, teniendo en cuenta lo señalado anteriormente.

```
Amstrad 128K Microcomputer    (v3)

 1985 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.
BASIC 1.1

Ready
Carlos Ruiz_
```

Ya hemos escrito todas las minúsculas en la pantalla. Vayamos ahora a escribir también las mayúsculas en orden alfabético. Para evitar mantener la pulsación de la tecla SHIFT, existe otra tecla auxiliar (CAPS LOCK), cuya pulsación provoca la representación mayúscula de las letras siguientes.



Si repite la pulsación de la tecla CAPS LOCK, volverán a aparecer las minúsculas.

LA BARRA ESPACIADORA

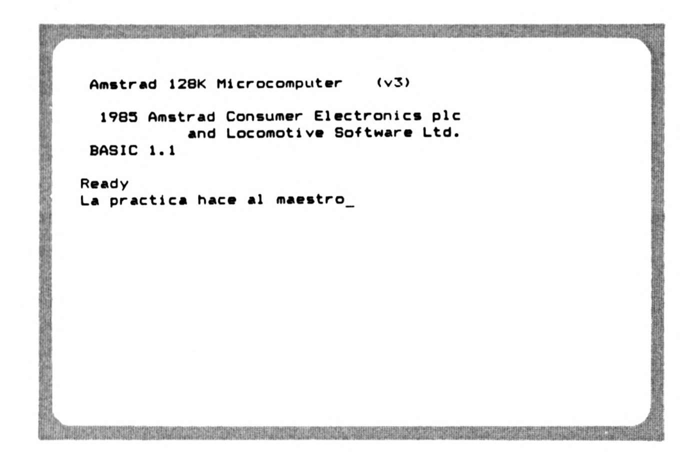


Esta tecla existe en cualquier máquina de escribir. Su ordenador también le permite contar con la posibilidad de insertar blancos entre las palabras. Si en la posición, donde debe aparecer un blanco, ya existe otro carácter, éste será borrado naturalmente.

Limpie la pantalla y teclee la frase siguiente:

La practica hace al maestro

En este caso debe pulsar la barra espaciadora entre todas las palabras.



LA TECLA DEL

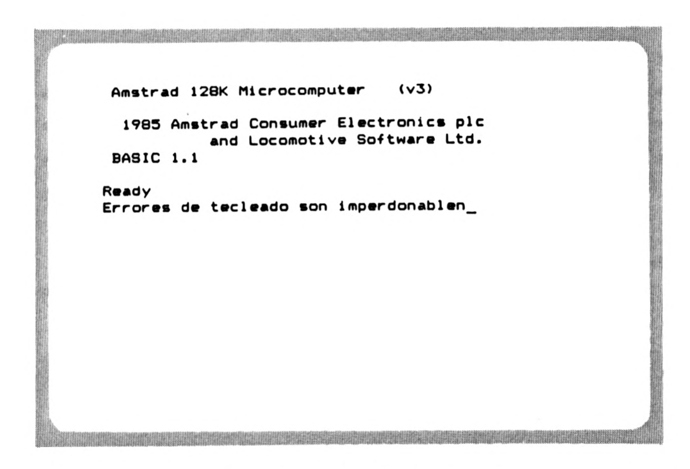


La abreviación de esta tecla ya permite adivinar su función. DEL significa DELETE (borrar). Con la tecla DEL podemos borrar el carácter situado a la izquierda del Cursor. Esta característica tiene especial utilidad al corregir errores de tecleado. Una simple pulsación de la tecla DEL, y el carácter erróneo ya desaparece.

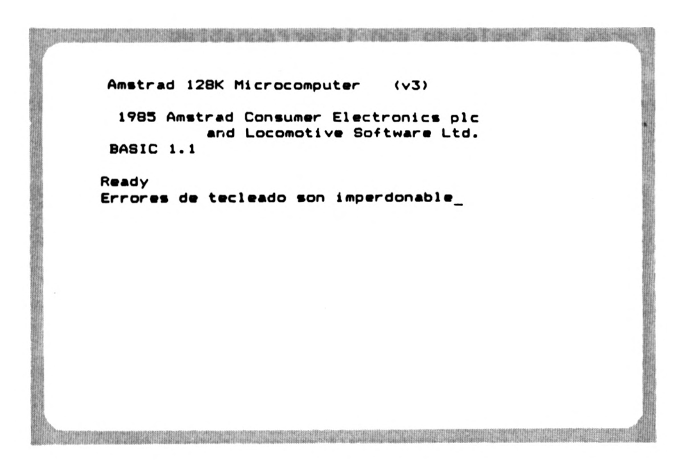
Ahora limpie la pantalla y escriba la frase siguiente:

Errores de tecleado son imperdonablen

Por supuesto, al efectuar la entrada de la última letra, hemos cometido un error.



En este caso no necesitamos utilizar Tipp-Ex para borrarla. La simple pulsación de una tecla elimina la última letra entrada. Pulse la tecla DEL.



Ahora puede sustituir aquí la letra correcta, y olvidarse del error de tecleado.

```
Amstrad 128K Microcomputer    (v3)

 1985 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.
BASIC 1.1

Ready
Errores de tecleado son imperdonables_
```

Sin embargo, este ejemplo no muestra la trascendental ventaja de la tecla. Ejecutemos otro ejemplo. Entre la frase siguiente tras haber limpiado la pantalla:

Errores de tecleado son imperrdonables

De nuevo se ha colado un error de tecleado: aquí sobra una 'r'.

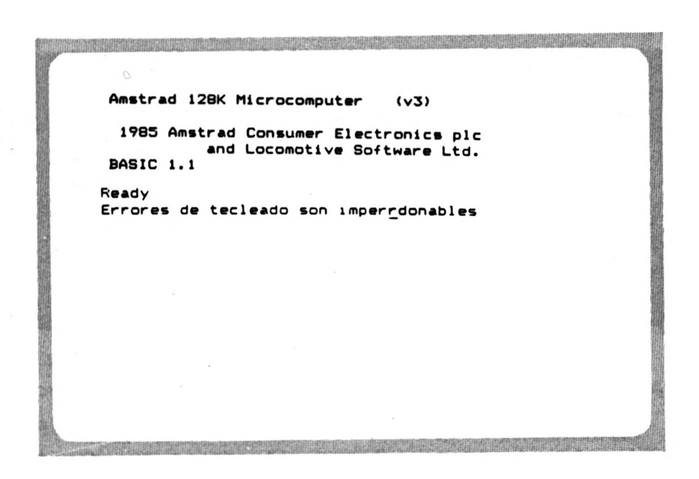
```
Amstrad 128K Microcomputer    (v3)

 1985 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.
BASIC 1.1

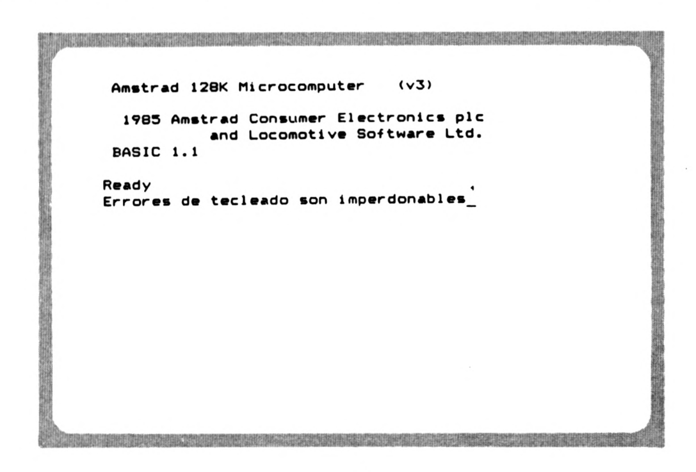
Ready
Errores de tecleado son imperrdonables_
```

No es necesario volver a escribir la última palabra a partir del error cometido, porque resulta, que al borrar el carácter situado a la izquierda del Cursor con la tecla DEL, todos los caracteres situados a la derecha del mismo van desplazándose hacia la izquierda, siguiendo su movimiento. Es un poco difícil explicarlo con palabras, por tanto véalo usted mismo.

Coloque el Cursor detrás de la 'r' errónea, utilizando la tecla CURSOR-IZQUIERDA.



Si ahora pulsa la tecla DEL, se borra la 'r' situada a la izquierda del Cursor, y el resto, incluyendo el carácter tapado por el cursor, se desplazando hacia la izquierda.

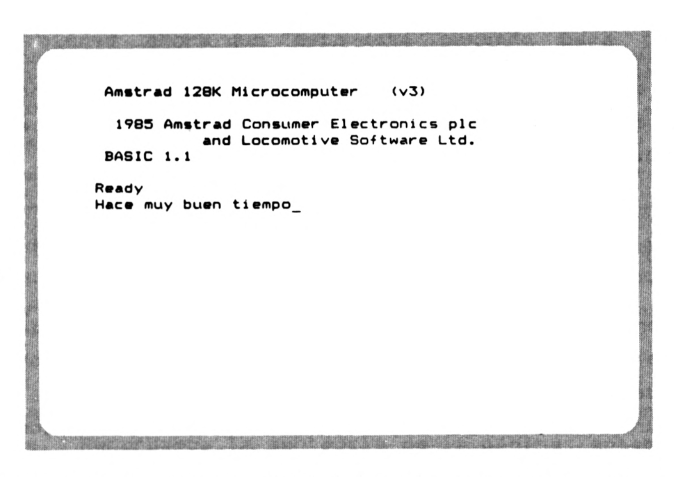


Si ahora conduce el Cursor al final de la frase, podrá continuar escribiendo. Para ello debe mantener la tecla CURSOR-DERECHA, hasta alcanzar el punto deseado.

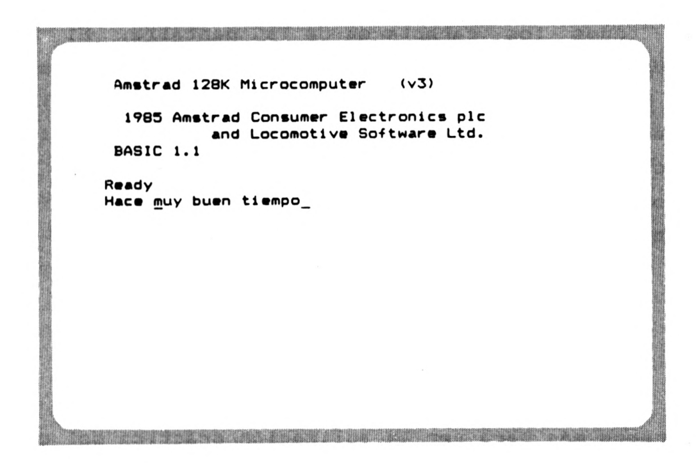
LA TECLA CLR



Tal como ya hemos dicho, la tecla DEL tiene la función de borrar el carácter situado a la izquierda del Cursor, así como juntar los demás caracteres situados a su derecha. El CPC dispone además de otra confortable ayuda de edición. La tecla CLR borra el carácter cubierto por el Cursor, desplazando con ello los caracteres situados a su derecha. Veamos otro ejemplo al respecto. Vuelva a escribir una frase errónea:



Ahora considera, que en realidad no hace tan buen tiempo como ha dicho. Quiere eliminar la palabra "muy". Cierto, podría borrarla con la tecla DEL, pero en este caso se trata de conocer la aplicación de la tecla CLR. Puesto que CLR borra el carácter cubierto por el Cursor, desplazando los caracteres de su derecha hacia la izquierda, debemos posicionarlo encima de la 'm' de "muy".



Pulsando cuatro veces la tecla CLR, se borrará el adverbio "muy".

Las teclas DEL y CLR constituyen, pues, dos importantes ayudas de edición, que le serán muy útiles a la hora de confeccionar programas o de efectuar cualquier otra entrada de texto.

DENSIDAD DE CARACTERES GRADUABLE

Seguramente ya habrá observado, que una línea de la pantalla contiene 40 caracteres. De esta forma podemos representar un total de 1000 caracteres, si contamos con 25 líneas. Muchos ordenadores de esta clase no se conforman con esta característica, e intentan por todos los medios representar 80 caracteres en lugar de 40 en cada línea. Ello exige una ampliación técnicamente laboriosa y consecuentemente costosa.

Los creadores del CPC tuvieron momentos muy lúcidos, cuando decidieron disponer una densidad de caracteres variable. De esta forma, el usuario puede escoger entre trabajar con 20, 40 e incluso 80 caracteres por línea, utilizando simplemente un comando BASIC. Ya conocemos un comando BASIC, el CLS, que elimina todos los caracteres de la pantalla. El comando utilizado para cambiar la densidad de caracteres se llama 'MODE'. Puesto que disponemos de tres opciones a elegir, hay que añadirle otra cifra al comando. Dicha cifra da información sobre el tipo de densidad, que se ha seleccionado. Por consiguiente: existen tres diferentes comandos MODE:

MODE 0 - cambiar a 20 caracteres por línea
MODE 1 - cambiar a 40 caracteres por línea
MODE 2 - cambiar a 80 caracteres por línea

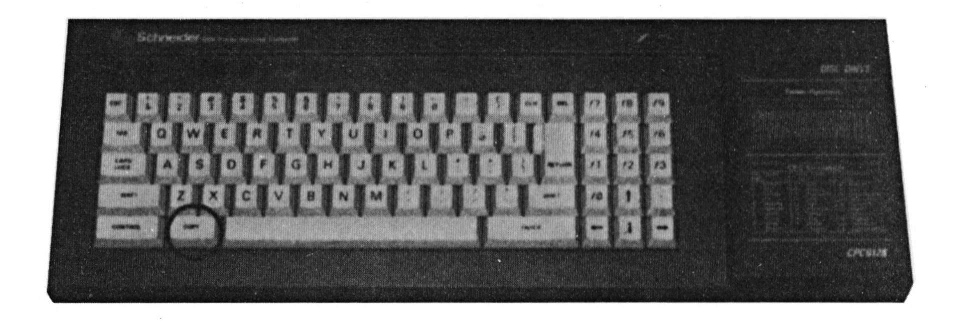
No esperemos más para ver la aplicación práctica del comando. Active su pantalla con el 'MODE 0'. Tenga en cuenta, que debe cerrar la sentencia con la tecla 'RETURN', para evitar una espera infructuoss a cualquier reacción del ordenador. Unicamente la tecla 'RETURN' puede provocar la ejecución de la sentencia.

Quizás piense ahora que "esto está pensado para personas miopes", si aparece el 'READY' hiperdimensional. Esta letra seguramente puede leerse con mayor facilidad, que la respresentada en el modo de 40 columnas. Se utiliza principalmente en programas de juego. Si acaso usted piensa utilizar esta letra p.ej. como titulares de un programa, aplicando una densidad diferente en las líneas restantes, debo decepcionarle lamentablemente. En una pantalla no puede utilizarse más que un único tipo de letra, ya que cualquier comando MODE empieza por limpiar toda la pantalla.

Veamos ahora la segunda alternativa, le representación de 80 columnas. Recordemos la sentencia de la página anterior, y entremos 'MODE 2', observando la reacción del ordenador.

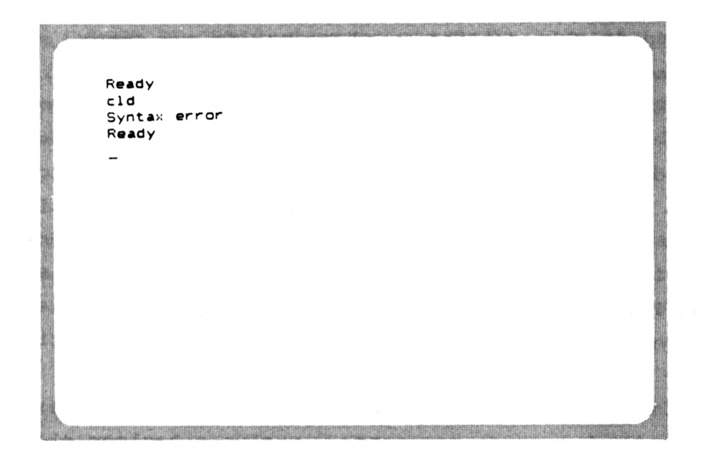
Si ahora escribe algo en la pantalla, observará, que no hay ninguna dificultad en leer los caracteres pequeños. Si usted no puede confirmarlo, seguramente posee un monitor de color, porque sólo el monitor verde ofrece una buena resolución de 80 columnas. Junto con el efecto protector de la vista, esta propiedad constituye la ventaja del monitor verde. Sin embargo, cuando haya visto el efecto del gran colorido del CPC, sabrá apreciar las ventajas de un monitor de color. Esto no quiere decir, que necesita dos monitores, aunque sería la solución más idónea. Una buena solución de compromiso supone la utilización de un monitor verde y un modulador HF, que pueda representar el colorido en un televisor normal y corriente.

EL CURSOR DE COPIA

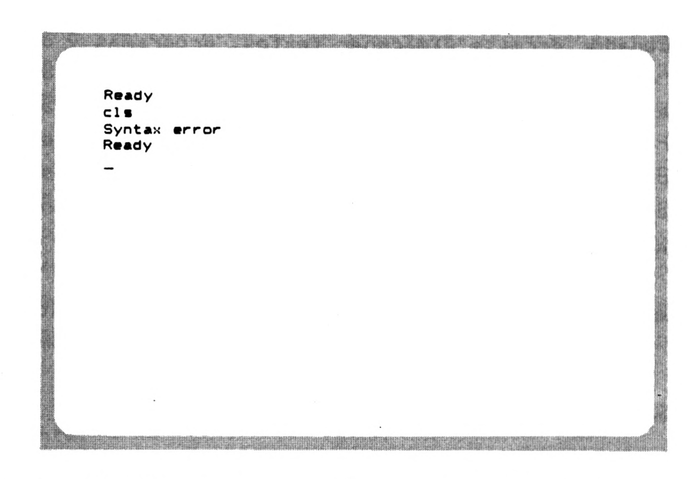


Todo ordenador permite almacenar textos, programas y datos, y volver a modificarlos en cualquier momento. En general, los programas no se escriben sólo una vez; con frecuencia deben efectuarse modificaciones del mismo. Ahora le enseñaremos la forma de modificar una secuencia de caracteres ya entrada, para facilitarle al máximo sus primeros intentos de programación.

Ejecutemos un ejemplo. Supongamos, que usted desee entrar el comando CLS. Sin embargo, al efectuar la entrada comete un error y teclea 'cld'. Por tanto, se ha equivocado al teclear la última letra. Ahora ejecute el ejemplo.



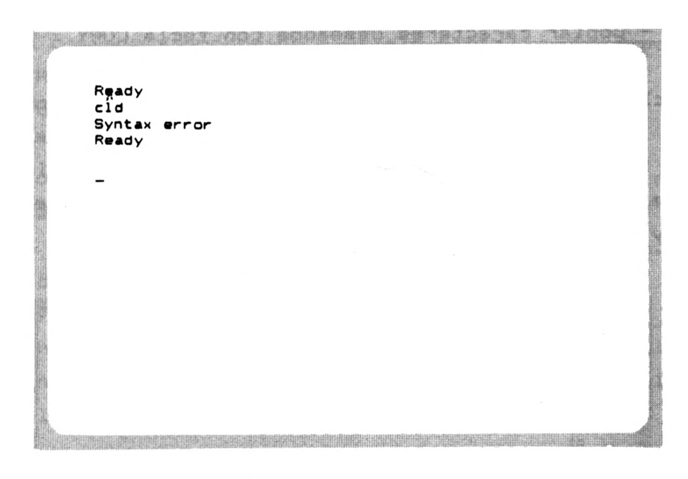
Pero ahora no queremos repetir la entrada del comando, sino modificar el indicado en la pantalla. Otros ordenadores de este tipo disponen de un llamado editor de pantalla. Con un editor de este tipo se puede llevar de nuevo el Cursor a la línea donde se halla el comando, modificarlo y volver a enviarlo con RETURN. Esto no es posible con el CPC. Inténtelo de todas formas. Lleve el cursor a la 'd' errónea y sustitúyala por una 's'. A la izquierda del Cursor vemos ahora el comando 'cls'. Si pulse la tecla RETURN, no se borrará la pantalla, como puede suponerse, sino que se emitirá de nuevo el mensaje de error.



La razón de ello estriba en el hecho, de que deben entrarse todas las letras que componen un comando antes de accionar la tecla RETURN. Es imposible completar o modificar caracteres que ya se hallan representados en la pantalla. Cuando se trata de líneas de instrucción muy largas, tales como las veremos más adelante, es un gran inconveniente repetir cada vez la línea completa. Sin embargo, el CPC le ofrece la posibilidad de evitarse todo este trabajo. Para ello se copian los caracteres necesarios de la línea "vieja" en otra posición de la pantalla. Se requiere un poco de práctica para dominar este método. Ahora le introduciremos paso a paso a dicha técnica. En primer lugar, limpie la pantalla -esta vez utilizando el comando correcto- y vuelva a teclear el comando erróneo 'cld'.

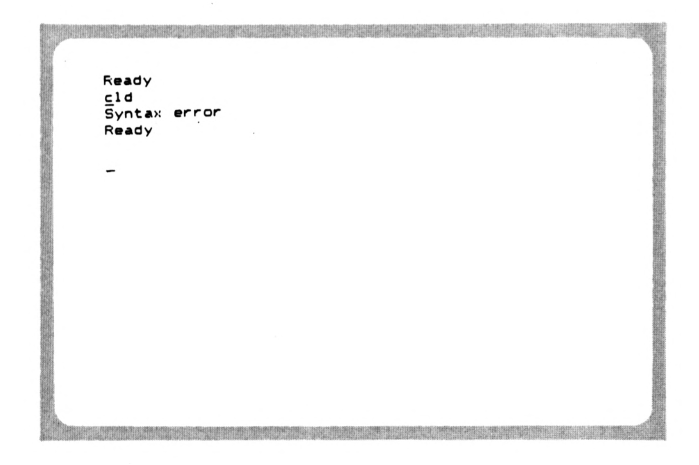
Paso 1 - CURSOR hacia el principio de la "nueva" línea

Empiece por decidir el lugar, en el cual debe copiarse la "vieja" línea de comando, y posicione el Cursor al principio de la línea deseada. En nuestro ejemplo, se trata de la línea número 6 de la pantalla.



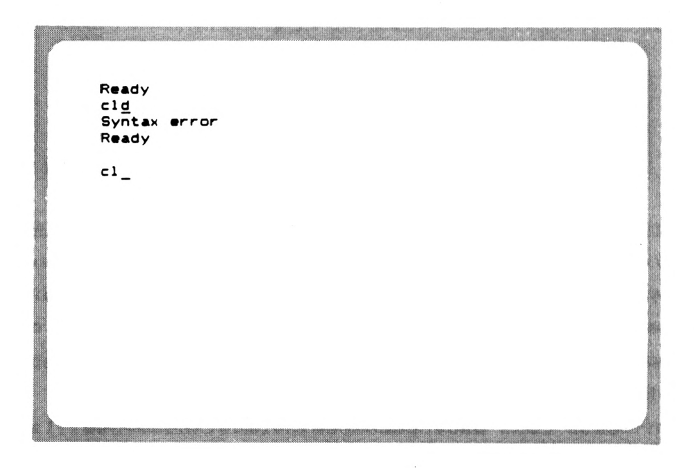
Paso 2 - CURSOR DE COPIA al principio de la "vieja" línea

Usted ya sabe manejar el Cursor descrito hasta ahora. ¿Pero qué es el Cursor de copia y cómo se mueve? El Cursor de copia es un segundo Cursor, cuyo aspecto es idéntico al del habitual. También es movido mediante las teclas de Cursor, pero en este caso debe pulsarse simultáneamente la tecla SHIFT. Mantenga ahora esta tecla, pulsando además la tecla de Cursor ARRIBA, hasta que el Cursor de copia haya alcanzado la "vieja" línea.

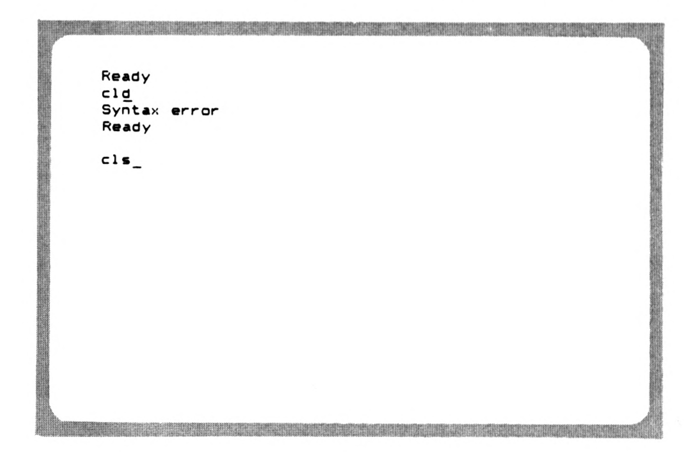


Paso 3 - Copiar caracteres deseados con tecla COPY

La tecla COPY mueve simultáneamente los dos Cursores -el normal y el de copia- hacia la derecha, copiando al mismo tiempo los caracteres cubiertos por el Cursor de copia. Esto suena más complicado de lo que es en realidad. Pruébelo usted mismo y pulse dos veces la tecla COPY.



Ahora ya ha copiado los dos primeros caracteres del comando CLS. La tercera letra no debe ser copiada, sino que debe ser escrita de nuevo. Por tanto, pulse ahora la tecla 's'.

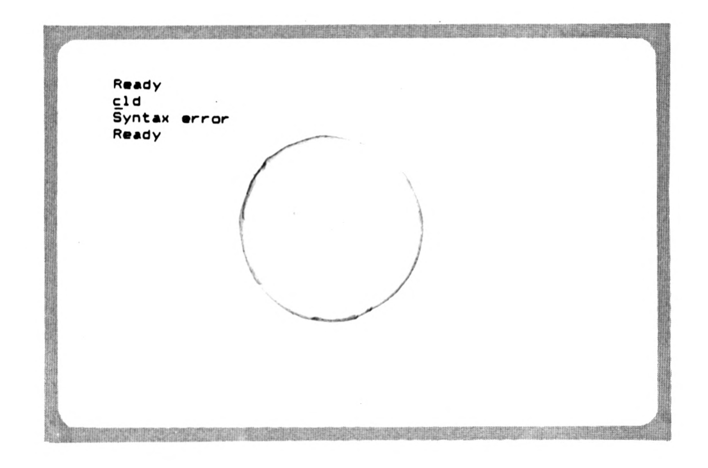


Ya hemos copiado y corregido el viejo comando. Ahora pulse la tecla 'RETURN', y se ejecutará el comando. Al mismo tiempo también desaparecerá el Cursor de copia.

Además de este método, existe otro diferente para cambiar líneas de comando existentes. Hemos copiado la "vieja" línea en otra posición de la pantalla y la hemos modificado. Otra posibilidad consiste en modificar la "vieja" línea en el mismo sitio. Antes ya fracasó nuestro intento de llevar simplemente el Cursor al lugar a rectificar. Ejecutemos ahora la segunda posibilidad paso a paso, después de volver a entrar el comando erróneo 'cld'

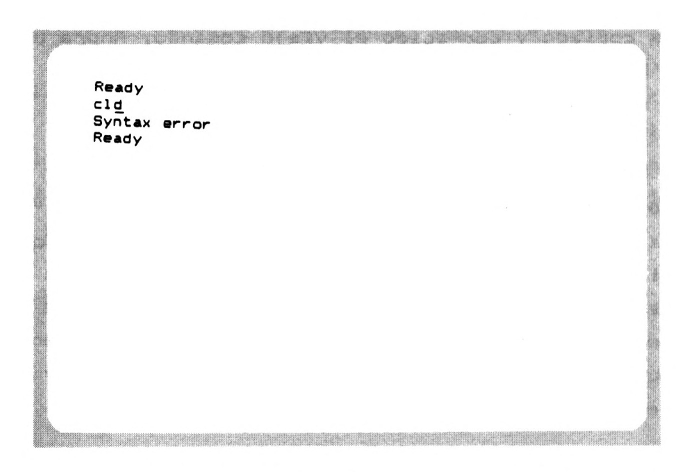
<u>Paso 1 - CURSOR al principio de la "vieja" línea</u>

Por tanto, no posicionamos el Cursor en un lugar diferente, sino al principio de la línea a modificar.

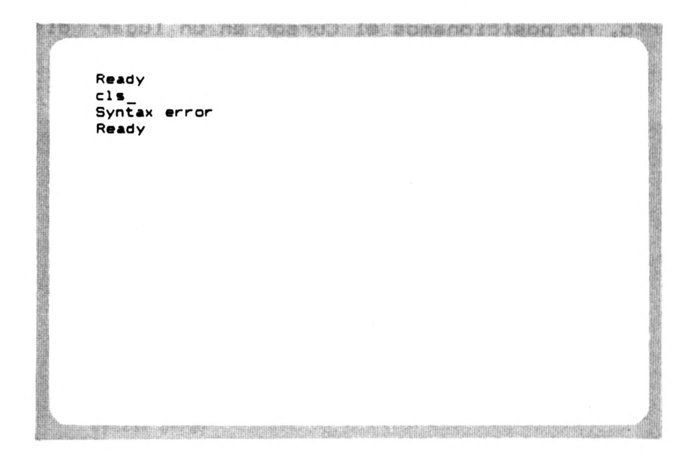


<u>Paso 2 - llevar tecla COPY al primer carácter erróneo</u>

Ahora accione dos veces la tecla COPY. Los dos primeros caracteres serán copiados en el mismo lugar, ya que las posiciones del Cursor normal y del Cursor de copia son idénticas. Por consiguiente, Usted debe pulsar la tecla COPY en lugar de Cursor hacia la derecha.



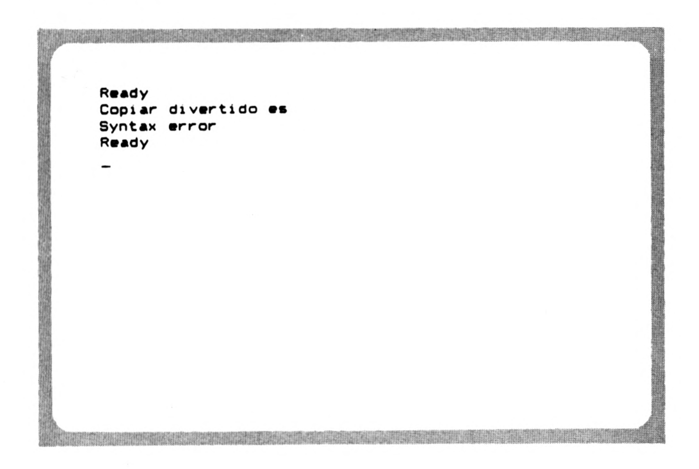
En este caso tampoco se copia el tercer carácter, sino que se sustituye por 's'. Por tanto, pulse la tecla 's'.



Ahora se puede ejecutar el comando correcto mediante la tecla 'RETURN'. Ya hemos visto dos posibilidades de modificar líneas de pantalla existentes, parece que el segundo método es el más sencillo. Usted debe decidir el método a utilizar según cada caso.

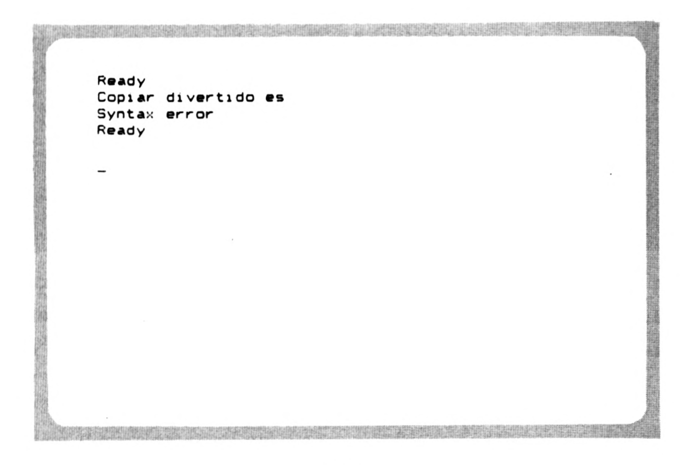
Posiblemente aún tenga dificultades a la hora de aplicar el Cursor de copia. En este caso, vamos a realizar otro ejemplo más completo. Escriba en primer lugar la siguiente secuencia

de palabras, después de haber limpiado la pantalla. A continuación, pulse RETURN.

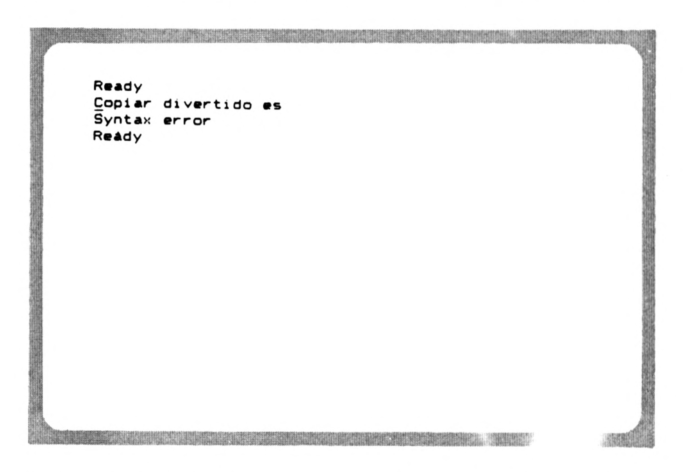


Ahora utilizaremos el Cursor de copia para convertir dicha secuencia en la frase "Copiar es divertido". Puesto que la secuencia no representa ninguna instrucción real, debemos aceptar el "Syntax error" que aparece después de accionar 'RETURN'. Pero ello no representa ningún obstáculo para conseguir nuestro objetivo.

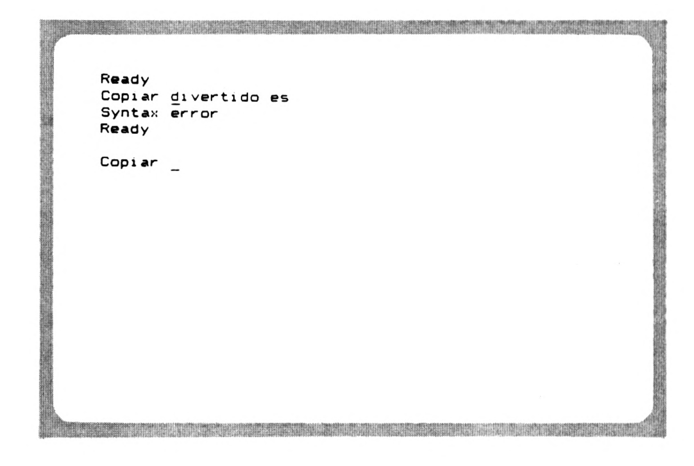
La frase correcta deberá aparecer en la línea 6 de la pantalla. Para ello mueva el Cursor al principio de dicha línea.



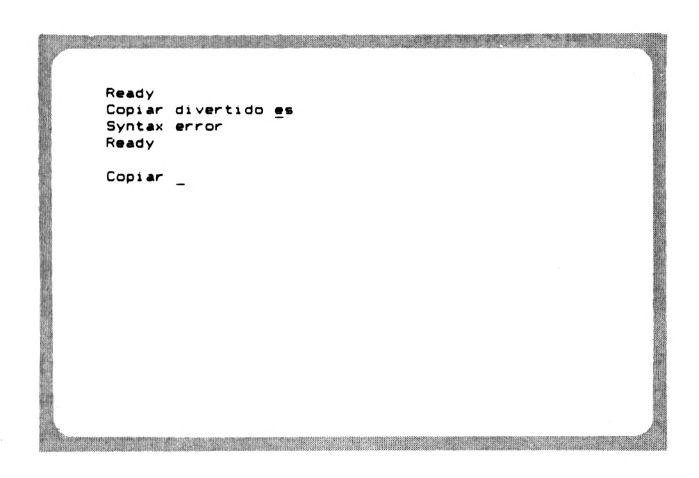
Ahora llevaremos el Cursor de copia al principio de la primera palabra a copiar. Mantenga pues la tecla SHIFT, y mueva el Cursor de copia con las teclas de Cursor.



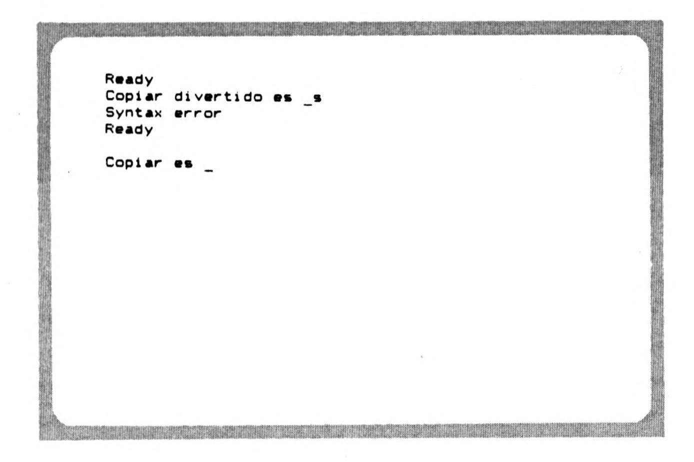
Ahora podemos copiar la primera palabra: pulse nuevamente la tecla COPY, hasta que el Cursor de copia coincida con la 'd' de la palabra 'divertido'. Por tanto, también debe copiarse el espacio.



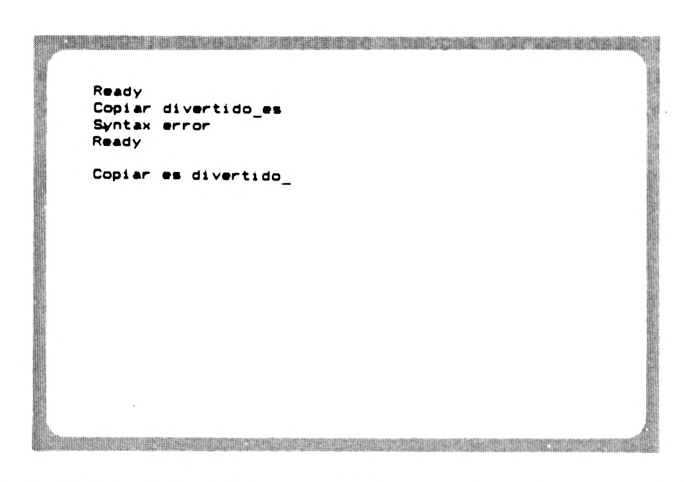
Ahora copiaremos la segunda palabra. LLeve el Cursor de copia al principio del verbo 'es', pulsando diez veces Cursor DERECHA.



A continuación copie el verbo, pulsando tres veces la tecla COPY. La última pulsación corresponde al espacio.



Ahora copie usted mismo la palabra "divertido". Mueva pues el Cursor de copia y pulse COPY. Después de haber conseguido el objetivo deseado, su pantalla ofrecerá el aspecto siguiente:



¿Lo ha conseguido? ¡Enhorabuena! Si realiza algunos ejercicos más, pronto dominará por completo esta técnica. Simplemente debe procurar no confundir el movimiento de ambos Cursores. De lo contrario pueden darse situaciones, que sólo podrá resolver con algo de rutina.

CAPITULO 2:

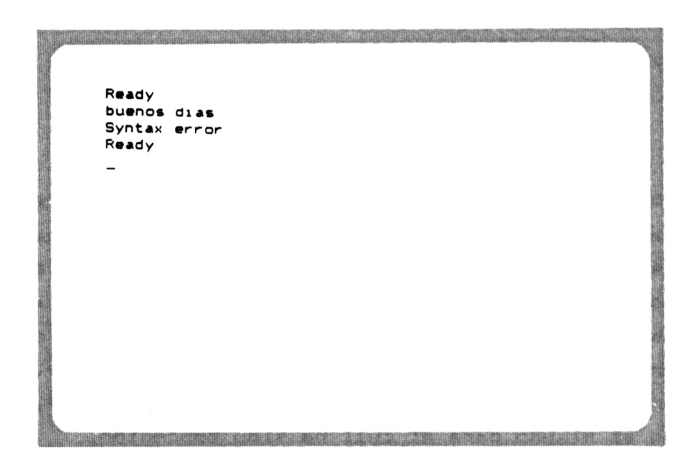
EL PRIMER COMANDO

Ya no basta con escribir cualquier tipo de caracteres en la pantalla, si queremos proseguir nuestras explicaciones sobre el teclado. Realmente deben ser pocos los lectores que hayan adquirido el CPC6128 con este propósito. Advertimos a todas aquellas personas, que se hayan aburrido en el capítulo anterior, que este capítulo será más interesante. Ahora utilizaremos al ordenador para ejecutar instrucciones, que en realidad es el uso para el cual ha sido concebido.

LA TECLA RETURN



RETURN es la tecla más importante del ordenador. Al pulsarla se transmite el texto entrado al ordenador, que interpreta la línea como instrucción e inicia los pasos correspondientes. Al finalizar la instrucción, el ordenador vuelve a presentarse con 'READY', igual que sucede al conectarlo. Por consiguiente, vuelve a estar dispuesto a recibir más instrucciones. Ahora intente saludar al ordenador. Entre el texto "BUENOS DIAS" y, a continuación, pulse la tecla RETURN. ¿Cuál será la respuesta del ordenador? Véalo Usted mismo.



Aquí aparece el primer mensaje de error de su CPC6128, el SYNTAX ERROR (error de sintaxis). El ordenador no puede entender la sintaxis, es decir, la combinación de caracteres. Es verdad, que actualmente existen ordenadores grandes, que pueden "conversar" con el hombre sobre temas limitados, pero este requiere un equipo de capacidad y precio mil veces superior. Todo lo que usted escriba en el teclado y "envíe" con la tecla RETURN, es analizado por el interpretador BASIC y ejecutado, si la entrada ha sido correcta.

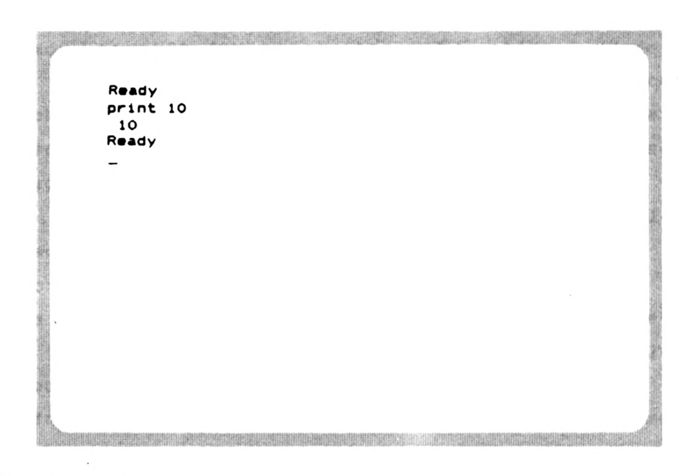
Su CPC6128 está equipado con el lenguaje de programación BASIC, y, por lo tanto, sólo puede entender instrucciones escritas en este lenguaje. El BASIC es el lenguaje de programación más extendido, que ofrece condiciones óptimas especialmente para los programadores inexpertos. Los comandos suelen derivarse del idioma inglés. ¿Cuánto tiempo precisaré para parender este lenguaje? Esta pregunta es muy frecuente, aunque no puede contestarse con seguridad. Depende de muchos factores. Por un lado, depende de la capacidad de aprendizaje del principiante, y por otro también influye el tiempo dedicado al ordenador. Hay personas, que dedican cada minuto libre al muy estimado ordenador. Ya no es extraño leer anuncios tales como "Vendó mi ordenador, amenaza de divorcio". En principio es necesario sacrificar un tiempo razonable, si pretende convertir la programación en su Hobby favorito. Si tiene conocimientos básicos de las matemáticas,

que comprendan incluso los principios del Algebra, sumado a un tiempo invertido de 10 horas semanales, ya pueden producirse los primeros resultados positivos después de unos tres meses. Es verdad, que la capacidad de razonamiento lógico y abstracto contribuye al éxito, pero ésta también irá desarrollándose continuamente de programa a programa. Además de los conocimientos de programación, es indispensable adquirir también experiencia para confeccionar programas exigentes, tales como p.ej. un pequeño procesador de textos o una gestíon de ficheros. La experiencia ha demostrado, que la calidad de estilo va mejorando de programa a programa. Le apuntaremos un pequeño dato: muchos principiantes ya disfrutan confeccionando pequeños juegos al cabo de tres meses de haber adquirido el ordenador. Mientras haya entusiasmo, usted también se habrá convertido pronto en uno de los numerosos programadores aficionados.

EL COMANDO PRINT

Es casi imposible imaginarse un programa sin el comando PRINT. Se encarga de cualquier salida en un programa. Tanto, si se trata de sacar un resultado matemático por la pantalla, o de enviar una dirección hacia el cassette, no hay nada que hacer sin este comando. Además, toda la emisión por impresora se realiza con el comando 'PRINT'.

De momento sólo trataremos el modo directo, es decir, que todavía no escribiremos programas. El modo directo significa, que usted entra un comando, lo cierra con RETURN y espera al resultado inmediato. Al principio lo ejecutaremos sin que se produzca ningún mensaje de error. Escriba pues el comando 'print 10', seguido de RETURN.



Por lo tanto, PRINT conduce todos los parámetros añadidos al comando hacia la pantalla (la emisión a dispositivos externos se efectúa con variaciones del PRINT). Los parámetros son componentes de un comando, que describen con precisión los efectos que debe causar el comando.
En nuestro ejemplo, el parámetro de salida era el número '10'. Por consiguiente, 'print 10' significa 'muestra el número 10 en la pantalla'. Naturalmente, además de números, también puede emitirse cualquier otro tipo de caracteres. En el transcurso de este capítulo podrá apreciar su gran variedad de aplicación.

CALCULAR CON PRINT

El día que quiera calcular la cantidad de impuestos que le devolverá Hacienda, y no dispone de calculadora en aquel momento, conecte simplemente el CPC6128. Yo mismo también enciendo con frecuencia el ordenador para realizar pequeños cálculos. Muchos amigos me preguntan entonces: "¿también se puede calcular con éso?". Y es que sería absurdo, que no se pudiesen efectuar cálculos con un ordenador, cuyo precio supera veinte veces al de una calculadora de bolsillo.

Pero vayamos al grano. Veamos primero las cuatro operaciones fundamentales de la aritmética. Para ello disponemos de cuatro símbolos en el teclado:

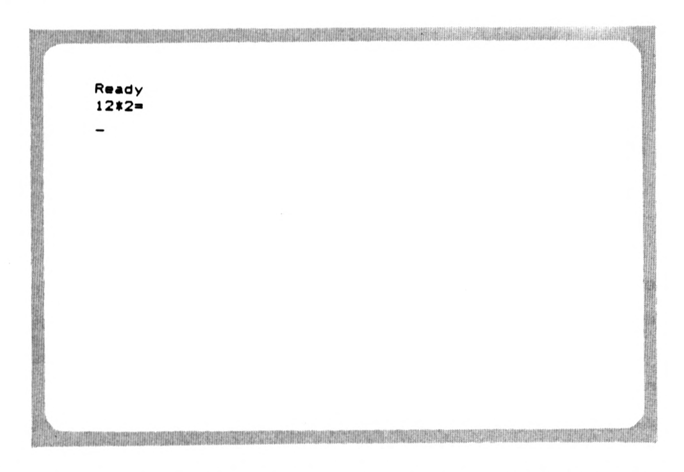
+ de adición

- de sustracción

* de multiplicación

/ de división

Pero los cálculos con su ordenador no pueden compararse con los de una calculadora. Por ejemplo, no puede entrar simplemnte '12*2=', ya que no sucede nada si no damos una instrucción. Siempre hay lectores amantes de los experimentos, que intentarán efectuarlo de todos modos. Lo que sucede entonces, es un poco confuso. Inténtelo usted mismo. Entre '2*12=' y envíe este "comando" con la tecla 'RETURN'.



Es difícil explicar lo que sucederá, sin anticiparnos a los hechos. El ordenador no muestra el resultado, sino que responde con READY. El Cursor salta al principio de la línea siguiente. ¿Pero qué sucede antes? El ordenador debe haber ejecutado alguna cosa, porque, de lo contrario, hubiera emitido un mensaje de error. No quiero anticiparme demasiado ahora. Pero debo dar unas cuantas explicaciones. Un programa está compuesto por varias líneas, que serán ejecutadas una tras otra después de iniciar el programa. Cada línea está señalada al principio con un número, que define el orden de su ejecución. En este caso, hemos entrado la línea 12 de contenido '*2='. No es una instrucción correcta, pero el ordenador no lo ha reconocido al entrarla, sino al ejecutar el programa. Pero vamos a ignorarlo de momento.

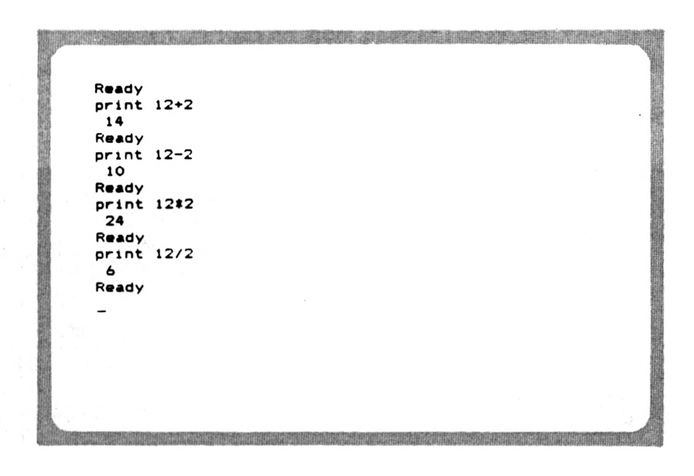
¿Cómo puede calcularse y emitirse ahora correctamente '12*2'? La solución es sencilla: entre la instrucción 'print 12*2'. Observe: siempre que entre una instrucción, debe cerrarla con RETURN. El ordenador pasa de lo que usted escribe en la pantalla. Sólo le interesa lo que le envía para su ejecución con RETURN.

```
Ready
print 12*2
 24
Ready
_
```

Quizás el resultado le sonsaque la primera exclamación "AH". Su ordenador ha realizado el primer trabajo para usted. Ha seguido fielmente su instrucción de multiplicar 12 por 2, y de mostrar el resultado en la pantalla.

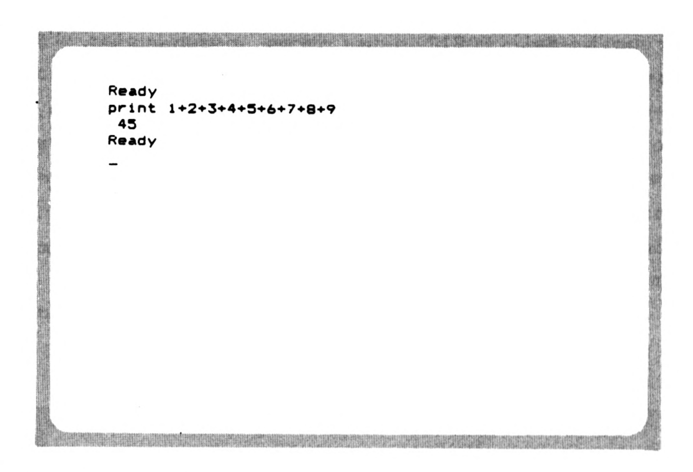
Aparte de todo esto quiero apuntar, que, de hecho, los lenguajes de programación sólo parecen ser tan complicados, porque hay que abreviar considerablemente las instrucciones para el ordenador. Cualquier lenguaje de programación debe estar organizado de forma clara hasta el último detalle. No existe, ni existirá, ningún lenguaje de programación, que acepte instrucciones tales como 'CALCULA 2*12 Y MUESTRA EL RESULTADO'. Olvídese de esta clase de ilusiones. La programación no es tan sencilla como eso. Cuando usted la domine, puede darse por satisfecho.

A diferencia del BASIC de otros ordenadores, aquí es imprescindible escribir un blanco detrás del comando PRINT, porque de lo contrario aparece el mensaje 'Syntax error'. Pero no es éste el único caso, sino que debe respetarse el espacio después de cualquier comando. Podrá aprenderlo rápidamente en el transcurso del libro. Pero ahora continuemos con las principales operaciones aritméticas. Calculemos ahora uno tras otro los resultados de '12+2', '12-2', '12*2' y '12/2'. Una vez efectuados, su pantalla debería presentar un aspecto parecido a éste:



¿Lo ha conseguido? Bien, entonces puede avanzar otro paso en el camino, que le convierte en programador aficionado. En caso negativo, le recomendamos que vuelva a leer con atención el último apartado.

Además de estos cálculos tan sencillos, también puede efectuar cálculos largos hasta un máximo de 255 caracteres. Sin embargo, es importante considerar la jerarquía de las operaciones: multiplicación y división se ejecutan siempre antes que la suma y la resta. Ahora póngale al ordenador el problema siguiente: '1+2+3+4+5+6+7+8+9'.



LAS OPERACIONES CON PARENTESIS

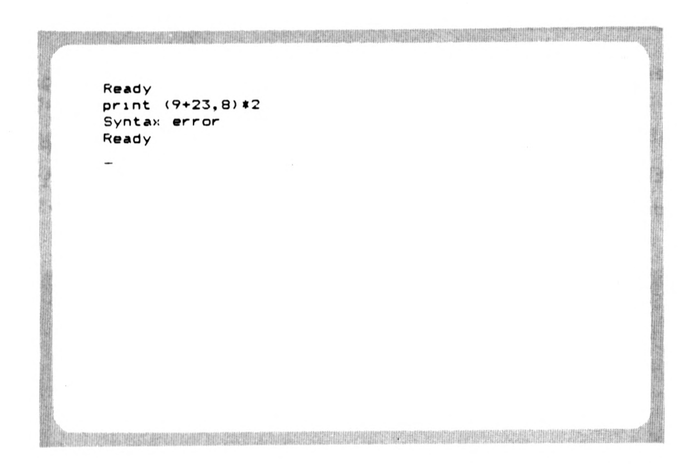


Ya hemos visto, que existe la posibilidad de efectuar cualquier tipo de operaciones encadenadas. Vea un ejemplo práctico: se trata de calcular el precio total de tres alfombras. El precio por metro cuadrado es de 1475 Ptas. La primera pieza tiene una superficie de 2.45 m * 2.80 m, la segunda de 4.50 m * 3.85 m y la tercera pieza mide 2.5 m * 4.80 m. Además hay que añadirle el 12 % de IVA. ¿Cuál es el precio total de de las tres piezas de alfombra? Si usted ha colocado los paréntesis correctamente, basta con una única instrucción PRINT:

```
print (2.45*2.8+4.5*3.85+2.5*4.8)*1475*1.12
```

Esto parece más complicado de lo que es en realidad. Dentro del paréntesis se calculan los metros cuadrados de alfombra. Después se multiplica el total por el precio por metro cuadrado, y finalmente se le añade el impuesto sobre el valor añadido. ¡Y todo ello con una única instrucción! Si usted ha entrado los datos correctamente, obtendrá un resultado de 62862.877, es decir un precio de 62.863 Ptas.

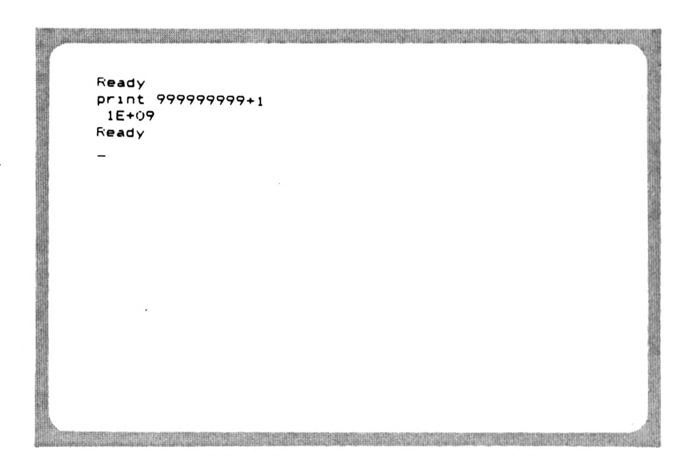
Como ha podido observar, en lugar de la coma decimal se utiliza un punto. Aunque ésta sea una norma americana, también es aplicada en los ordenadores españoles. Si usted entra la coma en relación con operaciones entre paréntesis, por ejemplo print (9+23,8)*2, aparece el mensaje de SYNTAX ERROR.



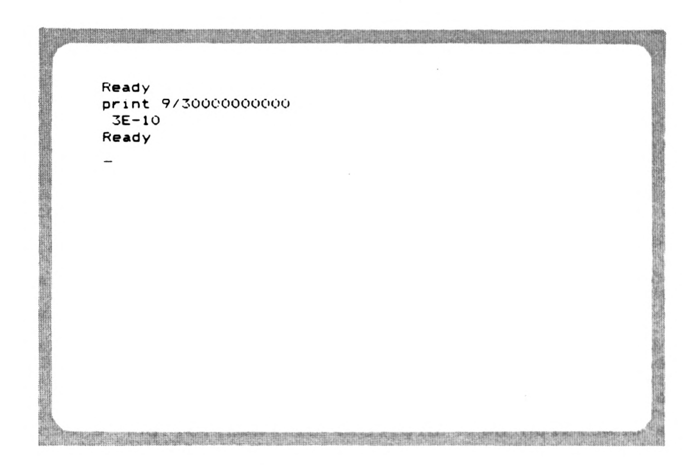
NOTACION EXPONENCIAL

El ordenador ha presentado el resultado con 5 decimales. Se trata de un número llamado de coma flotante. Siempre calcula con una precisión de 9 dígitos. Si resulta, por ejemplo, un número con 3 dígitos enteros, automáticamente calcula 6 decimales. Si se supera el número 999999999, también se vuelve a calcular con un precisión de nueve dígitos. Los números que superan dicha cantidad reciben un complemento, que indica la dimensión del resultado real. Entre por ejemplo la operación PRINT 999999999+1. El resultado será un número

de 10 dígitos, que no admite la representación habitual.



No desespere al ver este resultado. Simplemente significa, que da un resultado de '1 * 10 ^ 9, es decir un 1 con 9 ceros. 'E+09' significa base 10, exponente +9. Pero el exponente también puede ser negativo, tal como confirma el resultado de la operación 'PRINT 9/3000000000'.



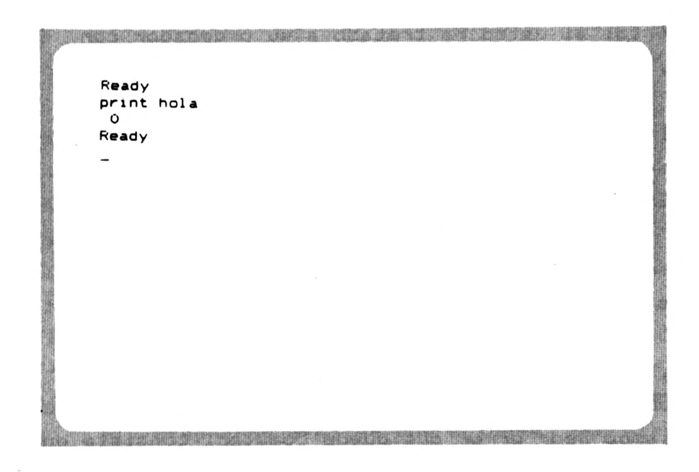
Aunque el resultado debería ser 0.0000000003, el ordenador utiliza la notación exponencial para expersarlo. Por tanto, '3E-10' equivale a '3 * 10 elevado a -10', lo cual signifca, que el número 3 se halla diez posiciones detrás de la coma decimal. De momento, esto es todo lo que usted debe saber

sobre los cálculos en modo directo con las cuatro operaciones aritméticas fundamentales. En su manual del CPC6128 encontrará otras funciones matemáticas.

EMISION DE TEXTO CON PRINT



Además de números, con PRINT también pueden emitirse textos, los llamados **Strings** (variables de cadena). Intentos precipitados tales como PRINT HOLA fracasan irremediablemente.



Aunque la instrucción no registra el resultado deseado, no aparece un mensaje de error. Por consiguiente, el ordenador ha ejecutado la instrucción. ¿Pero qué es lo que ha ejecutado? ¿Qué significa el cero? Este tipo de problemas son habituales a la hora de realizar los primeros pasos con el

BASIC. Es difícil contestar a estas preguntas sin anticiparnos en el tema. En este caso se indica el contenido de una variable (memoria numérica interna). El nombre de la memoria es HOLA, y se emite el valor 0 porque no hemos depositado nada en la variable HOLA. No hay razón para desesperarse, si no acaba de entender esta explicación. Más adelante explicaremos con detalle el tema de las variables.

Pero volvamos a la cuestión de emitir variables de cadena (en lo sucesivo continuaremos utilizando la denominación técnica de **Strings**), o sea de emitir Strings con el comando PRINT. La solución es muy sencilla: los Strings están enmarcados entre comillas. Aplicando esto a nuestro ejemplo, volvemos a escribir la instrucción de la forma siguiente:

```
PRINT "HOLA"
```

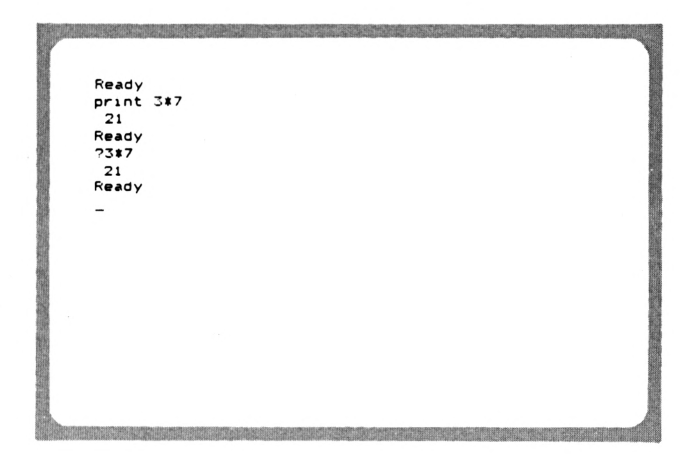
Ahora, después del primer intento fracasado, hemos conseguido finalmente nuestro objetivo. El ordenador le desea que pase un buen día (ya sabemos, que hace todo lo que usted le ordene).

Por cierto: si usted ha tratado a su ordenador con excesivo cuidado, pensándose tres veces cada instrucción antes de dársela para evitar causarle algún daño, puedo tranquilizarlo. Ninguna instrucción, por muy disparatada que sea, puede provocar, que su ordenador desprenda una nube de humo. Existen pocas aficiones, para las cuales la frase "se aprende de errores" sea tan acertada como en ésta. Un aficionado a la electrónica seguramente se estirará los cabellos, si efectúa una soldadura incorrecta de un terminal de 3000 Ptas. con un amplificador de producción propia. Sin embargo, si usted causa un error en su programa de 32 KBytes, puede corregirlo en cuestión de cinco minutos. El único, y al mismo tiempo peor error que puede cometer, consiste en llevar trabajando p.ej. ocho horas seguidas en la elaboración de un programa, y que su madre enchufe la aspiradora defectuosa. En este caso, un cortocircuito tiene efectos desastrosos: la pérdida de ocho horas de trabajo.

También es posible destruir p. ej. una cassette importante y valiosa por su trato indebido. Pero observando las reglas más importantes sobre el almacenamiento de datos, no puede fallar nada.

ENTRADA PRINT SIMPLIFICADA

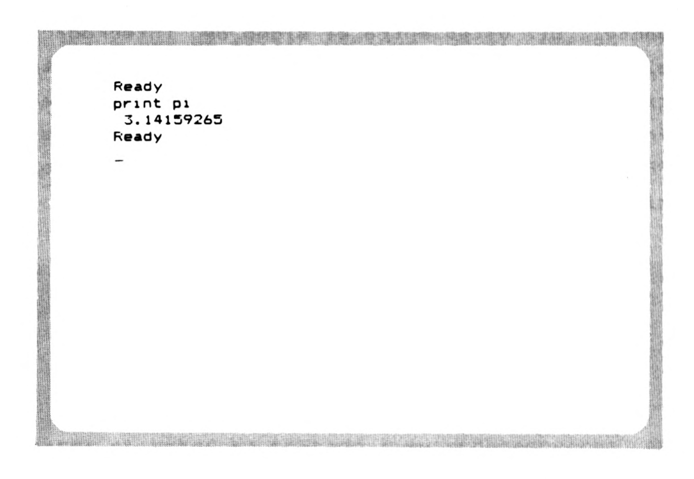
El comando PRINT también puede ser sustituído por un interrogante. Si usted entra el signo de interrogación en lugar del comando PRINT, puede ahorrarse un poco de trabajo, en especial, porque PRINT suele ser el comando utilizado con mayor frecuencia. Por consiguiente, usted puede escribir tanto 'PRINT 3*7' ó '? 3*7'. Pruébelo. No es necesario dejar un espacio detrás del interrogante.



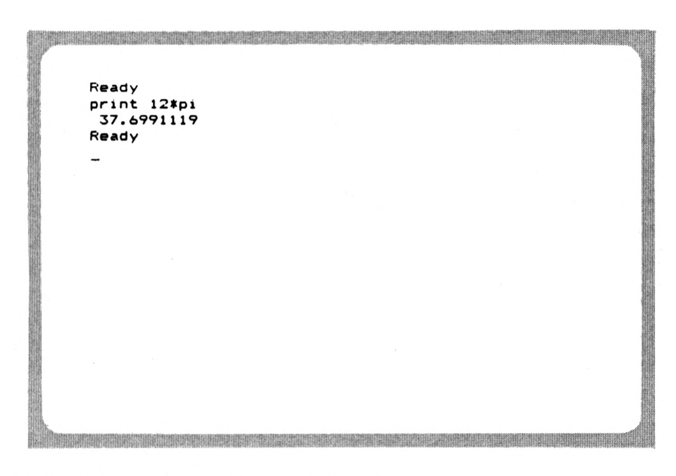
Ahora depende de usted el tipo de comando PRINT que vaya a utilizar. La abreviación en forma de interrogante resulta muy práctica cuando se trata de efectuar algún cálculo breve. En ese caso, el ordenador efectúa operaciones con la misma rapidez que una calculadora de bolsillo.

PI Y POTENCIACION

Volvamos ahora a las matemáticas. Todos conocemos la constante PI, utilizada en los cálculos de circunferencias y esferas. ¿Recuerda el valor de PI? Era 3.1413 ó 3.1214 ó ...? No se mate. Normalmente se calcula sólo con el valor 3.14, o sea, con una precisión de dos decimales. El CPC6128 ya contiene la constante PI. Compruébelo, sacándola a la pantalla con el comando 'PRINT PI'.



¡PI con una precisión de ocho decimales! Es más que suficiente. Ahora calcule el perímetro de un círculo de diámetro 12 cm. Utilice la fórmula: perímetro = diámetro * PI.

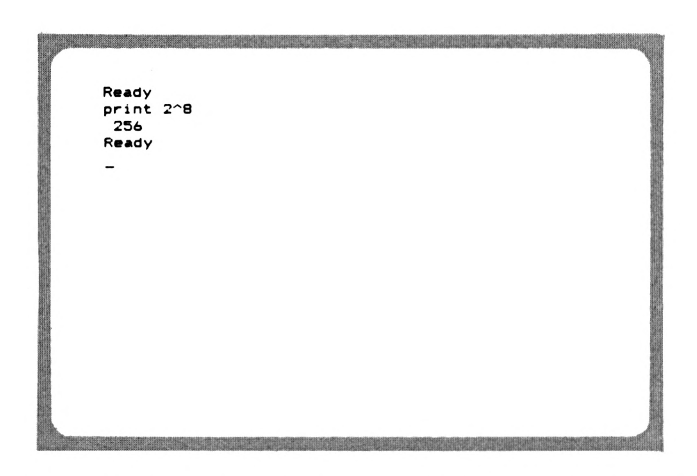


Por tanto, el perímetro es de 37.7 cm.

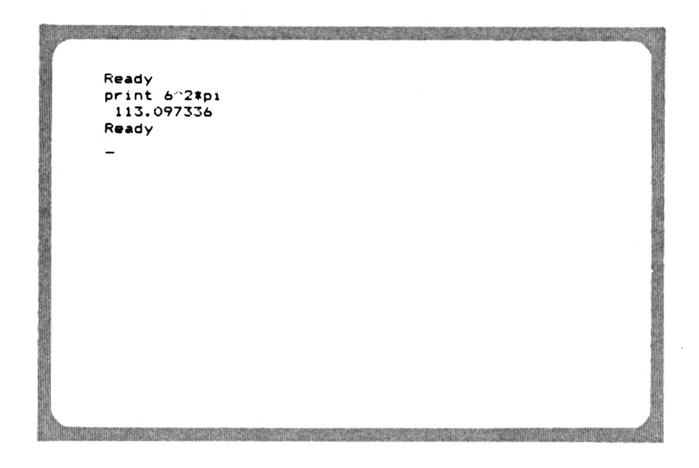
Veamos ahora la potenciación. Potenciación significa multiplicar un número por él mismo. El exponente define el número de veces que debe multiplicarse la base por si misma.

2 elevado a 3 significa 2*2*2, o sea 8
10 elevado a 4 significa 10*10*10*10, o sea 10000

En el primer ejemplo, la base es '2' y el exponente '3'. Pero dejemos ahora las lecciones de matemáticas. Para expresar una potencia en el CPC6128, se coloca una 'flecha ascendente' entre la base y el exponente. Pruébelo calculando 2 elevado a 8.

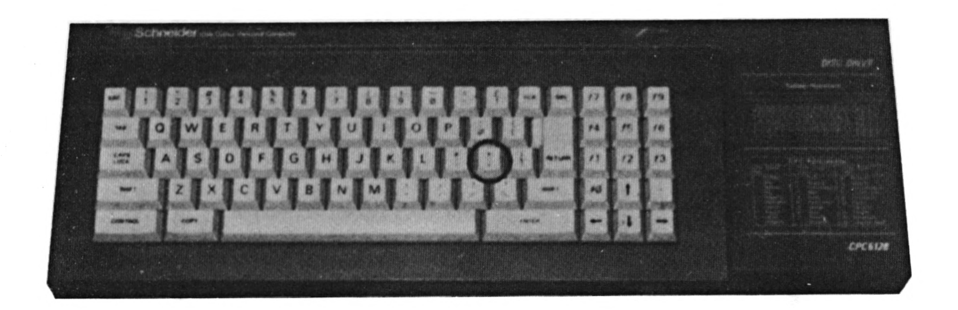


Esto no ha sido difícil. Ahora calcule la superficie de un círculo de diámetro 12 cm. La fórmula a aplicar es: superficie = PI por radio al cuadrado (el radio es medio diámetro).

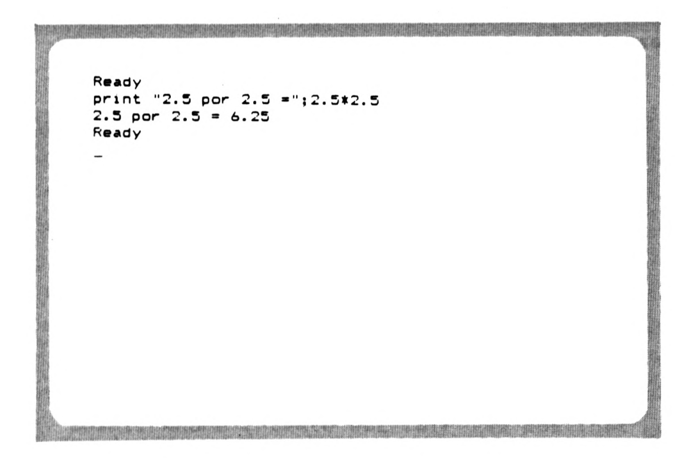


El resultado da una superficie de unos 113 cm^2. Pero acabemos ya con las matemáticas. Volvamos a centrar nuestra atención en cosas más interesantes.

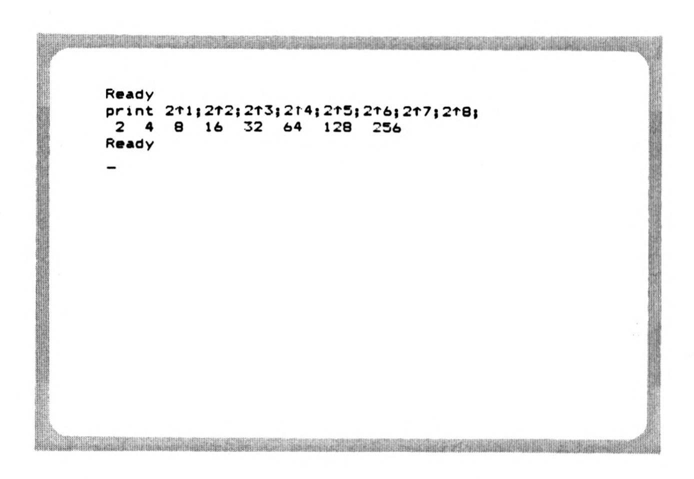
COMBINAR STRINGS CON NUMEROS



Cuando se trata de combinar Strings y números, las teclas ';' y ',' adquieren gran importancia. ¿Pero en qué sentido? Por ejemplo, si Vd desea emitir una línea tal como "2.5 por 2.5 = 6.25", que además debe calcular el resultado al mismo tiempo, se precisa un String seguido de una operación matemática. El resultado será el siguiente:



Por consiguiente, el punto y coma separa Strings de números. Pero no sólo eso; Los Strings y números u operaciones pueden separarse entre si mediante el punto y coma. De esta forma podemos efectuar p.ej. varias operaciones con un comando PRINT y mostrar los resultados uno al lado de otro. Intente ahora hallar las potencias del número '2', desde el exponente '1' hasta el '8', y mostrar los resultados en una línea.



Una instrucción PRINT seguida de PUNTO Y COMA tiene el efecto de que la siguiente instrucción PRINT no se emita en la línea siguiente, sino en la misma, detrás del primer PRINT. Más adelante le explicaremos más detalles sobre el tema.

Escriba de nuevo la última instrucción PRINT, sustituyendo esta vez el punto y coma por una coma.

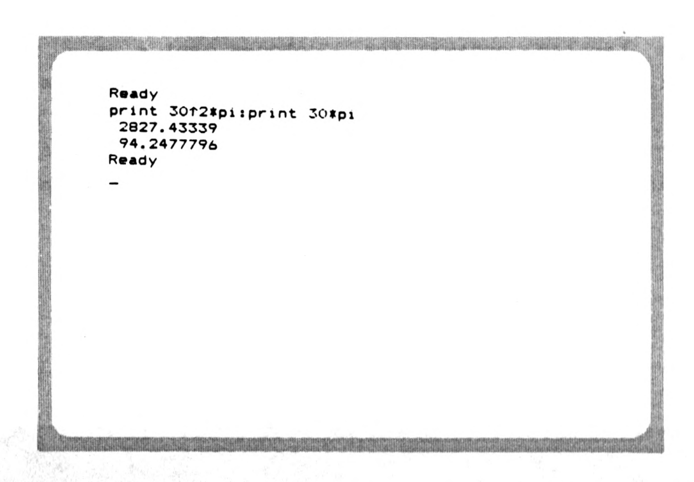
```
print 2↑1,2↑2,2↑3,2↑4,2↑5,2↑6,2↑7,2↑8
 2             4             8
 16            32            64
 128           256
Ready
_
```

La coma también separa los valores entre si, pero a distancias mayores. Si se dedica a contar los espacios entre ellos, verá que se trata de 13 caracteres. Ello permite realizar una emisión de números ligeramente formateada. Para la separación de Strings, en cambio, no suele utilizarse la coma.

SEPARAR INSTRUCCIONES

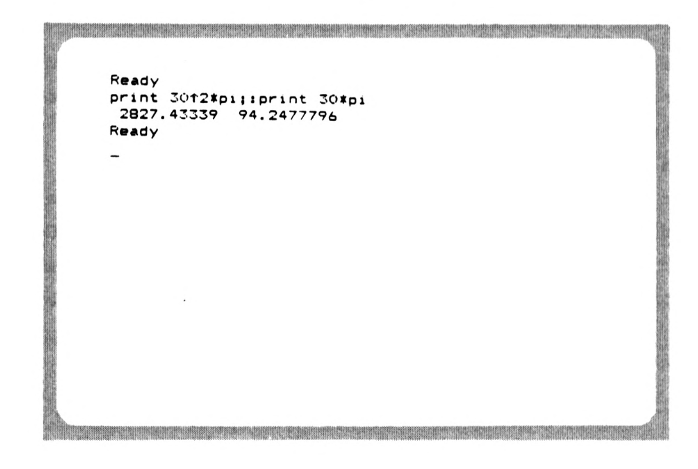


Una línea de instrucción puede estar formada por un máximo de 255 caracteres. Las instrucciones BASIC no suelen alcanzar esta extensión. La separación entre instrucciones se efectúa mediante el signo de dos puntos, que permite escribir varias instrucciones seguidas en una única línea de programa o una línea de entrada directa. Veamos un ejemplo. Queremos efectuar dos operaciones con un único RETURN. Los resultados deben aparecer uno debajo de otro. Nuestro ejemplo calculará 30^2*PI y 30*PI. Ahora intente escribir las dos instrucciones PRINT en una única línea, separadas por dos puntos.



Naturalmente también podemos escribir tres o cuatro instrucciones separadas por dos puntos. Sólo debemos tener en cuenta, que no superen el límite de 255 caracteres.

Aquí también podemos mostrar claramente el efecto del punto y coma, que sigue a la primera instrucción PRINT. Limpie la pantalla y vuelva a escribir las dos instrucciones PRINT anteriores, separándolas con un punto y coma.



El punto y coma después del primer PRINT provoca, que esta línea no sea cerrada todavía. El segundo PRINT sigue inmediatamente al primero. Los espacios en blanco se deben al hecho de que el ordenador reserva delante de la primera cifra un espacio para el signo. Sin embargo, un número positivo no será señalado con '+', sino únicamente el número negativo va precedido del correspondiente signo (-).

Tenga en cuenta, que el espacio detrás de los dos puntos no tiene ningún significado. Tanto si aquí escribimos uno, cinco o ningún espacio, no influye sobre el resultado.

CAPITULO 3:

EL PRIMER PROGRAMA

Al decir primer programa no nos referimos a TV1, sino a sus primeros pasos en la programación BASIC. No tema tocar este "hierro candente". El BASIC es un lenguaje para el principiante, que no debe infundir demasiado respeto a nadie. El hecho de que en las tiendas especializadas en la venta de ordenadores suelen encontrarse niños, que pasan el tiempo con el BASIC después de salir del colegio, debería acabar de desvanecer por completo sus últimos escrúpulos. No se limite eternamente a gastar su tiempo con los programas confeccionados. Sea sincero, ¿no ha pensado algunas veces en confeccionar sus propios programas? Este capítulo y el siguiente deben ayudarle a dar un pequeño paso hacia el gran mundo interesante de la programación.

QUE ES UN PROGRAMA?

En el lenguaje técnico, un programa se define como una "secuencia de instrucciones para resolver un problema determinado". Por lo tanto, la secuencia de instrucciones, ordenadas de forma lógica, compone un programa. Eso significa, que, para obtener un programa se parte del planteamiento del problema y se formula la secuencia lógica de instruciones. Lógicamente correcto significa, que, además de conocer todos los comandos del BASIC, también es necesario combinarlos de forma racional para llegar a resolver el problema planteado. Así, por ejemplo, es inútil conocer infinidad de vocablos ingleses, si no sabemos combinarlos correctamente para poder establecer una comunicación con otra persona.

LA NUMERACION DE LINEAS

Hemos visto, que un programa es una secuencia de instrucciones. ¿Pero cómo se define su orden de ejecución? Veamos, en el lenguaje de programación BASIC cada instrucción (también llamada "Statement") recibe un número, que define el orden en que se ejecutará el programa. Imagínese p.ej. un programa almacenado en su cerebro, que debe resolver la operación aritmética '48/12' con ayuda de la calculadora. Se dará cuenta, de que en este caso también hay que seguir cierto orden de ejecución:

1. Busca calculadora.
2. Conecta calculadora.
3. Si no hay pilas, vé a paso 12.
4. Pulsa la tecla '4'.
5. Pulsa la tecla '8'.
6. Pulsa el signo de división.
7. Pulsa la tecla '1'.
8. Pulsa la tecla '2'.
9. Pulsa la tecla '='.
10. Lee el resultado.
11. Almacena resultado en la memoria.
12. Desconecta calculadora.

Es sorprendente, la cantidad de pasos en que puede dividirse un proceso aparentemente tan sencillo. Como puede ver, todos los pasos están numerados para definir el orden de su ejecución. Pero ésta no es la única razón. El paso 3 plantea una importante lógica de programación: el comando de salto. Si se cumple una condición determinada (¿no hay pilas?), el programa se bifurca hacia el paso 12. Este número, es la llamada dirección del comando. Por consiguiente, sin este número sería imposible dirigirnos a una instrucción determinada.

En el capítulo anterior ya hemos visto el comando PRINT, que sirve para emitir datos en la pantalla. Sin embargo, sólo lo hemos aplicado en modo directo, es decir, el comando ha sido

ejecutado inmediatamente después del RETURN. Ahora vamos a codificar una secuencia de tres instrucciones PRINT en forma de programa. ¿Qué debemos hacer? Correcto, cada instrucción recibe un número de línea, que define el orden de ejecución del programa.

El CPC6128 admite números de línea comprendidos entre 0 y 65535. La amplitud de paso puede ser cualquiera.

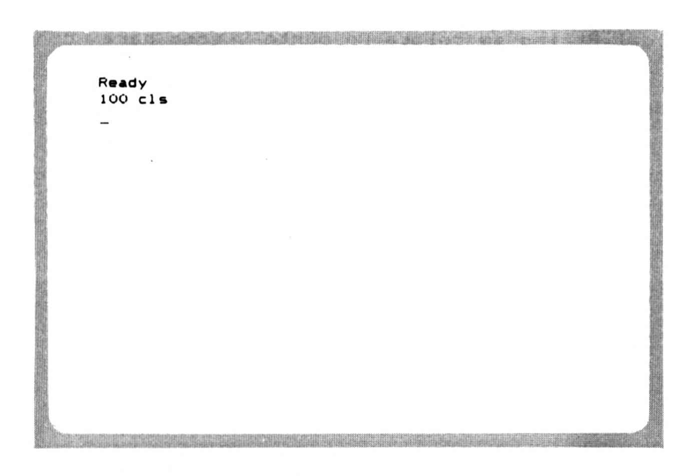
La amplitud de paso define el valor en que se incrementa el número de la línea anterior, para obtener el número de la línea siguiente. Así, por ejemplo, un programa de cuatro líneas puede estar formado por las líneas número 1, 8, 10, 20. También se admite la secuencia 100, 200, 300, 400. Por lo tanto, la amplitud de paso puede variar dentro de un mismo programa.

¿Pero qué motivo justifica una amplitud de paso mayor que 1? Está claro: si hay que insertar otra línea entre dos líneas existentes de un programa, la amplitud de paso que las separa debe ser superior a 1, puesto que los números de línea siempre son enteros. Por consiguiente, es imposible añadir otra línea entre las líneas 11 y 12. En cambio, sí disponemos de varias posibilidades para insertar otra línea entre 11 y 15. En la práctica, una amplitud de paso de 10 ha dado muy buenos resultados.

Pero volvamos ahora a nuestro problema, que se compone de los siguientes pasos:

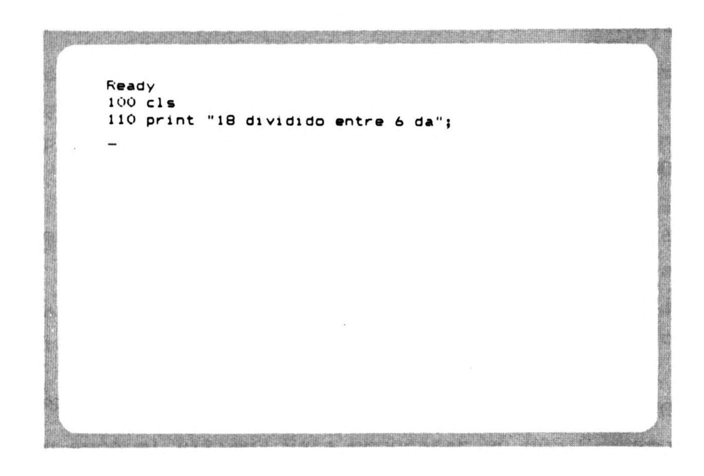
1. Limpia la pantalla.
2. Emite el texto "18 DIVIDIDO ENTRE 6 DA"
3. Emite el resultado justo detrás del texto.

Utilizaremos tres líneas para resolver el problema. ¿Cómo hay que plantearse ahora el problema? En primer lugar, debemos elegir el primer número de línea. En nuestro ejemplo será la línea 100. Para ello tecleamos pues el número 100, seguido del comando que limpia la pantalla. Recordemos: para limpiar la pantalla debemos utilizar el comando 'CLS'. Por lo tanto, la línea quedará como sigue:



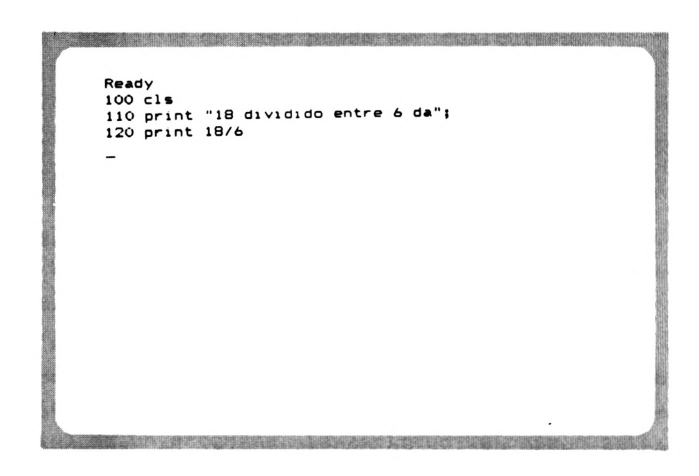
Detrás de cada número de línea debe figurar un espacio en blanco.

Esta es pues la primera línea de nuestro programa. La segunda línea debe emitir una cadena de caracteres. Ya sabemos, que ello se consigue con el comando PRINT. Trabajamos con una amplitud de paso de 10. En consecuencia, la segunda línea obtiene el número 110. Ahora escriba la línea.



Detrás del String se coloca un punto y coma para conseguir, que el siguiente PRINT sea emitido en la misma línea, es decir, detrás del texto.

Ahora intente escribir usted mismo la tercera línea. Se trata de calcular 18/6 y emitir el resultado.



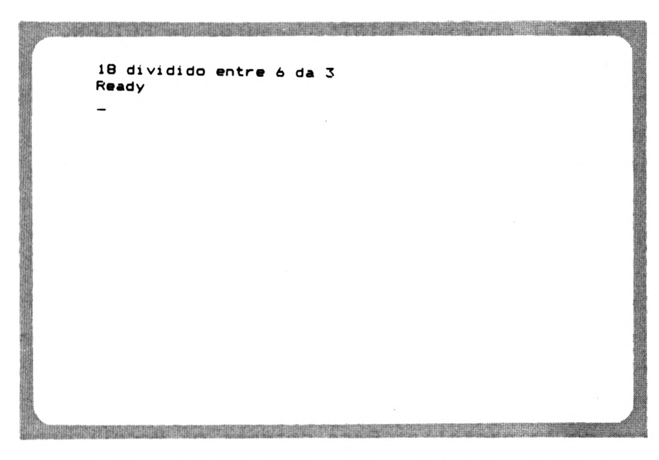
Ahora ya hemos terminado el programa, y sólo queda ejecutarlo.

EJECUCION DEL PROGRAMA

El CPC6128 no sólo guarda en la pantalla las líneas que acabamos de entrar, sino también en su memoria BASIC. Aunque usted limpie la pantalla ahora, el programa no se pierde. Puede volver a ejecutarlo en cualquier momento.

El comando 'RUN' inicia la ejecución de un programa BASIC

Ahora teclee el comando 'RUN' y observe lo que sucede en la pantalla.



Este programa representa un primer éxito para el principiante.

Ahora podrá ejecutarlo tantas veces como quiera. Compruébelo usted mismo: escriba otra vez 'RUN' y otra vez y ....

MODIFICACION DEL PROGRAMA

Puede darse el caso de tener que efectuar algunos cambios en el programa. Supongamos, que en lugar de 18/6 desee calcular 24/6 con el programa anterior. Para ello no es necesario volver a teclear las tres líneas, sino que puede modificar las líneas existentes en el lugar adecuado. Pero para ello debemos listar en primer lugar el programa anterior.

El comando 'LIST' muestra las lineas del programa en la pantalla.

"Otro comando nuevo", pensará usted. Y podemos asegurarle, que no será el último. Para programar en BASIC, debe estar familiarizado con los comandos más importantes.

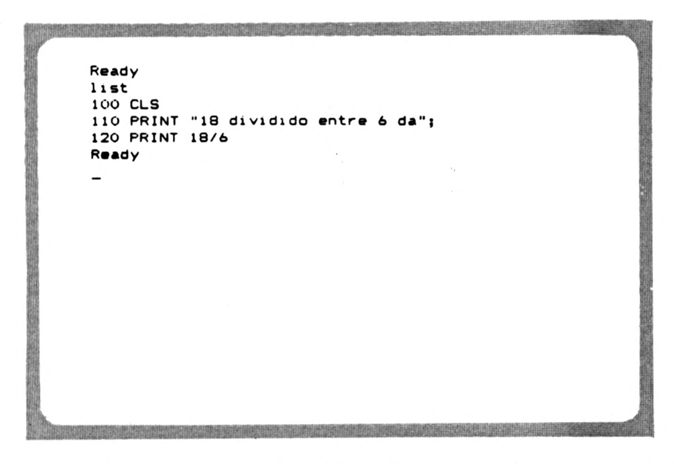
Ahora escriba el comando 'LIST', seguido de RETURN, y observe lo que sucede. En la pantalla aparecen las líneas del programa que acaba de entrar. Seguramente habrá notado, que el ordenador ha convertido los comandos en letra mayúscula. Con ello se pretende obtener un listado más claro. Por cierto, la denominación de 'listado' hace referencia a un programa emitido en pantalla o en impresora.

Debería saber otra cosa: cuando se trata de listar programas más extensos, se efectúa un "Scroll" de la pantalla. Esto significa, que se van desplazando las líneas en sentido ascendente, desapareciendo al mismo tiempo las líneas superiores de la pantalla. Para detener el Scroll debe pulsar la tecla 'ESC'. La pulsación de cualquier otra tecla reanuda el proceso del listado. Al pulsar 'ESC' dos veces seguidas, se interrumpe el listado.

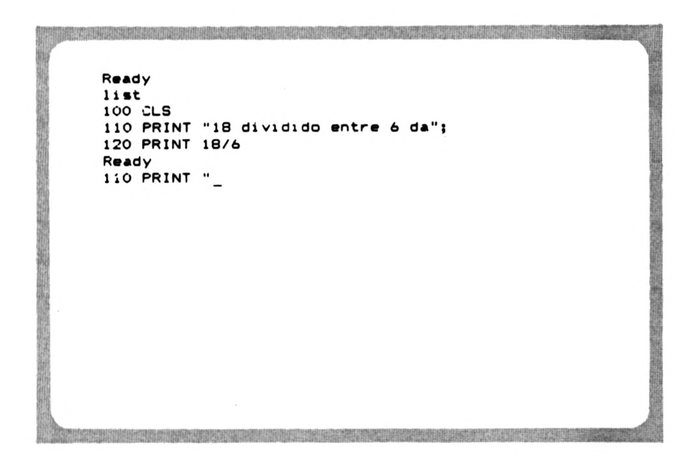
Volvamos ahora a la modificación del programa. Tenemos dos posibilidades de cambiar las líneas para calcular 24/6:

1. Utilizando el Cursor de copia.
2. Utilizando el comando EDIT.

El método del CURSOR DE COPIA ya ha sido explicado al principio del libro. Sin embargo, volveremos a explicarlo de nuevo, aplicándolo al caso que nos ocupa. La línea a cambiar se halla en el pantalla. Con ayuda de la tecla SHIFT, mueva ahora el CURSOR DE COPIA al principio de la misma:



Ahora pulse la tecla COPY hasta llegar al primer carácter a modificar:

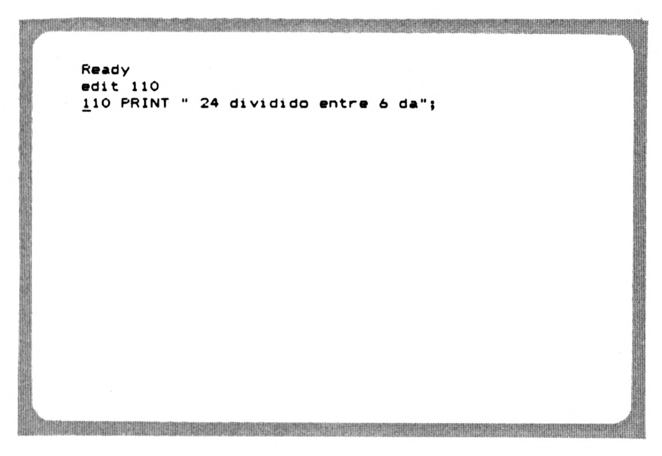


En este punto se efectúa el primer cambio de la línea. Ahora escriba el número '24'. Para evitar, que una nueva pulsación de la tecla COPY acepte también el '18', coloque el CURSOR DE COPIA sobre el blanco detrás del mismo número (¡no se olvide de SHIFT!).

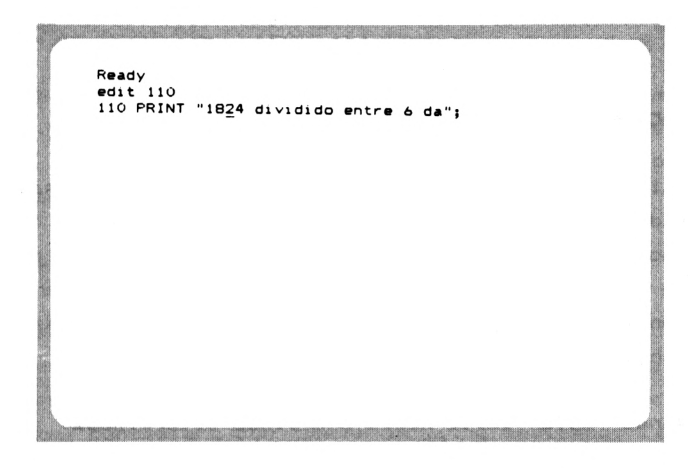
```
Ready
list
100 CLS
110 PRINT "18 dividido entre 6 da";
120 PRINT 18/6
Ready
110 PRINT "24_
```

Ahora copie el resto de la línea con ayuda de la tecla COPY, pulse RETURN, y aquí termina la modificación de dicha línea. Si queremos obtener un resultado correcto, también debemos cambiar la línea 120. Para ello siga los mismos pasos que en la línea 110. Ya sabemos, que no es fácil dominar el método del Cursor de COPIA, pero hay que practicarlo. Yo mismo tuve problemas para conseguir cierto dominio de este extraño método de edición, pero ahora no quisiera tener que prescindir de él.

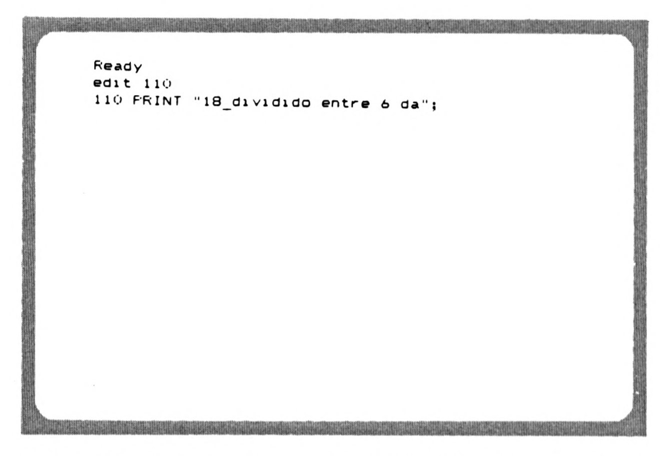
Veamos ahora el segundo método para modificar el programa. El comando 'EDIT' lleva una línea determinada a la pantalla para proceder a su modificación. Escriba la instrucción 'EDIT 110' (¡no se olvide de RETURN!):



Se muestra la línea 110, y el Cursor se encuentra al principio de la misma. Ahora coloque el Cursor sobre el primer carácter a cambiar. En este ejemplo queremos sustituir '24' de nuevo por '18'. Al escribir ahora '18', observará lo siguiente:



El número '18' no ha sido escrito sobre el '24', sino que ha sido insertado. El comando 'EDIT' siempre trabaja en modo de inserción. Por consiguiente, hay que borrar el número '24'. Utilice la tecla 'CLR', que, como recordamos, borra el carácter cubierto por el Cursor. Pulse esta tecla dos veces:



Una vez finalizadas todas las modificaciones, pulse la tecla RETURN. Verifique usted mismo, si la línea ha sido cambiada, escribiendo el comando 'LIST'.

Ya hemos visto dos métodos para cambiar un programa. Decida usted mismo el método que vaya a utilizar.

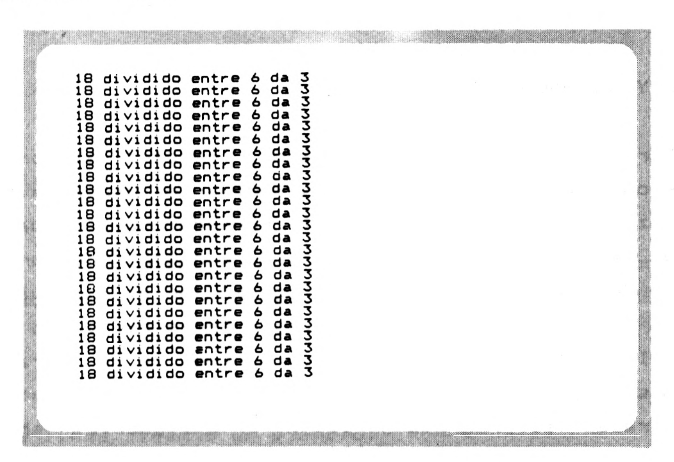
BIFURCACION

En realidad, es poco frecuente, que los programas sean ejecutados estrictamente desde el principio hasta el final, tal como sucede en el ejemplo anterior. Con frecuencia, casi siempre cumpliendo determinadas condiciones, se saltan líneas y se continúa la ejecución en otro punto. En estos casos, se utilizan los saltos.

El comando 'GOTO' salta hacia una línea determinada

El BASIC cuenta con el comando 'GOTO' para saltar hacia otra línea.

Vamos a ponerlo en práctica inmediatamente. Sabemos, que el programa finaliza después de la línea 120. ¿Qué ocurrirá si después, por ejemplo en la línea 130, damos un salto volviendo a la línea 110? Yo sostengo, que se emitirá ininterrumpidamente "18 DIVIDIDO ENTRE 6 DA 3". Pero véalo usted mismo. Escriba la línea '130 GOTO 110' y ejecute el programa con 'RUN'.



Con ayuda de la línea 130 hemos creado un bucle interminable, puesto que el programa ejecuta continuamente las líneas 110 110 a 130. Si no hacemos nada para evitarlo, el programa continuará así durante horas y horas.

La tecla 'ESC' interrumpe la ejecución de un programa

Si pulsamos ahora la tecla 'ESC', el programa se detiene. Cualquier otra tecla hará reanudar su ejecución. Si desea finalizar el programa, pulse la teclca 'ESC' dos veces. El ordenador contestará con "BREAK IN ...", indicando la línea en la cual se ha producido la interrupción del programa. Ahora bien, la línea, en la cual se ha detenido el programa, sólo viene determinada por la casualidad.

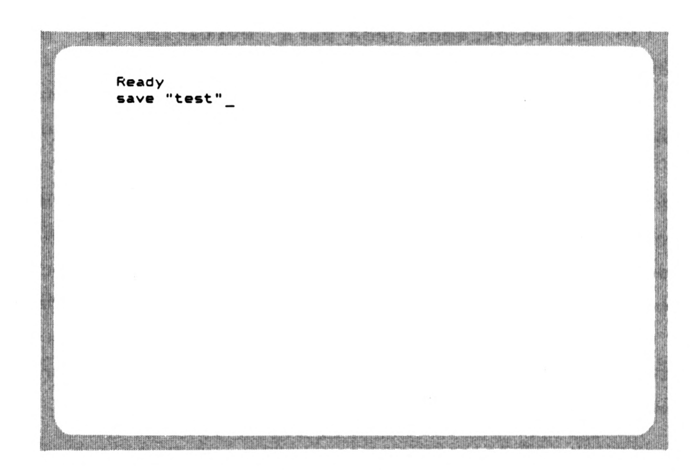
Si queremos reanudar la ejecución, disponemos de otro comando para ello. Continuar en inglés se llama 'continue". Puesto que los comandos del BASIC se derivan del inglés, el comando correspondiente se escribe 'CONT'.

El comando 'CONT' reanuda la ejecución de un programa tras su interrupción

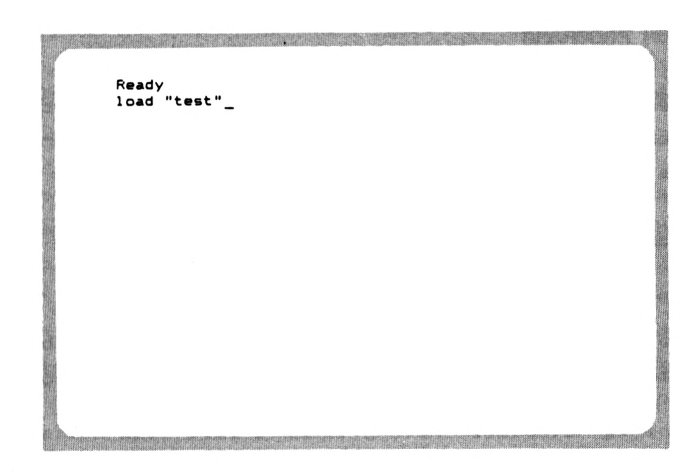
Escriba ahora este comando. El programa se encuentra de nuevo dentro del bucle interminable, y puede ser interrumpido o finalizado con ayuda de la tecla 'ESC'.

GRABAR Y CARGAR PROGRAMAS

Si ahora desconecta el ordenador, perderá el programa. Suponemos, que no quiere volver a teclear el programa deseado cada vez que vuelva a conectar el ordenador por lo cual puede guardar los programas en una unidad de disco incorporada al mismo. Grabemos ahora nuestro programa (en el capítulo 8 se explica el formateado de diskettes). Ahora elija un nombre para el programa, formado por un máximo de 8 caracteres. Elegimos, por ejemplo, el nombre "TEST". El comando utilizado para grabar el programa se llama 'SAVE'. Escriba ahora la instrucción correspondiente:



Si ahora desea volver a cargar y ejecutar el programa, utilice el comando LOAD.



En el caso de que usted no recordase el nombre del programa, puede visualizar el contenido del diskette en la pantalla con ayuda del comando **CAT**.

El programa se ejecuta con RUN, al cual puede añadirse un número de línea. Veamos un ejemplo:

Instrucción	**Función**
RUN	inicia un programa que se encuentra en la memoria
RUN 100	inicia un programa existente en la memoria, comenzando por la línea 100

Por consiguiente, el programa no es iniciado en la primera línea, tal como hemos visto hasta ahora, sino por la línea indicada en la instrucción.

Bueno, de momento, esto ha sido toda la información en lo que hace referencia a la grabación y carga con la unidad de disco. Más adelante, en el capítulo 'LA UNIDAD DE DISCO', insistiremos en otras posibilidades de dicha unidad.

BORRAR UN PROGRAMA

Ya sabemos, que el programa se borra al desconectar el ordenador. Pero ésta no es la única posibilidad de borrar un programa.

El comando 'NEW' borra el programa que se encuentra en la memoria

Si ahora escribe el comando 'NEW', el programa queda borrado. Naturalmente es aconsejable grabar previamente el programa a borrar.

A todos los lectores incrédulos: si entra el comando 'LIST' después de haber efectuado 'NEW', podrá comprobar, que el programa ha sido borrado efectivamente. Ya no se indica ninguna línea, y el comando 'RUN' no surte ningún efecto.

Ahora ha adquirido los primeros conocimientos básicos para la confección de un programa, lo cual le servirá de base para trabajar y entender los capítulos siguientes, un poco más complicados. Si el presente capítulo le ha presentado algún problema, le aconsejamos que vuelva a leerlo atentamente.

CAPITULO 4:

AYUDAS PARA LA PROGRAMACION

El extenso BASIC de su CPC6128, del que puede estar realmente orgulloso, es la razón que nos indujo a escribir el presente capítulo. No todos los ordenadores disponen en su versión estándar de comandos que facilite considerablemente la creación y elaboración de programas. Existen comandos para borrar líneas determinadas, para la numeración automática de líneas, ... mencionando tan sólo dos de las ayudas para la programación descritas a continuación. Cualquier persona, que quiera disfrutar de la versión BASIC 1.0 al confeccionar sus programas, no debería dejar de leer este capítulo.

NUMERACION AUTOMATICA DE LINEAS

Al crear un programa, la amplitud de paso suele ser 10. Al cabo del tiempo, puede ser bastante pesado escribir el número al principio de cada línea. Así también lo consideraron los creadores del BASIC utilizado por su ordenador, razón por la cual crearon un cómodo comando para la numeración automática de las líneas. Veamos, en primer lugar, la descripción de este comando.

Problema: **Numeración automática de líneas**

Instrucción: **AUTO** inicio, incremento

Parámetros: inicio - **línea inical**
incremento - **amplitud de paso**

Ejemplo: **AUTO 100,10**
Empezando por la línea 100, todas las líneas siguientes serán señaladas con una amplitud de paso de 10 (100, 110, 120, 130, ...)

Observación: **Con al tecla 'ESC' se puede interrumpir la numeración automática. Si la línea a entrar ya existe anteriormente , esto será señalado con un asterisco (*) detrás de dicho número. Con RETURN se conserva esta línea.**

Acaba de ver una descripción exhaustiva de un comando, tal y como podrá hallarlas en el transcurso del libro cada vez que se explique un nuevo comando. Aparece un término extraño, que de momento no significará nada para usted: parámetros. Con el nombre de parámetros se designa todo aquello que acompaña al comando, o sea, todo lo que se le añade. Así, por ejemplo, hay que añadir un parámetro al ya conocido comando 'LOAD', que es el nombre del programa a cargar.

Pero volvamos al comando AUTO. Supongamos que quiere entrar las diez primeras líneas de un programa, empezando por la línea 10 y dando pasos de 5 en 5. Habitualmente, al principio de cada línea se indica su número correspondiente. Por consiguiente, la primera línea recibe el número 10, la segunda el 15, y así sucesivamente. Como ya hemos dicho, el comando AUTO le evitará este trabajo. Ahora pruebe la instrucción, escribiendo:

```
AUTO 10,5
```

Inmediatamente se indica la primera línea (10). El ordenador espera ahora la entrada de dicha línea. Entre aquí por ejemplo 'PRINT "linea 10"' y finalice la entrada con RETURN. A continuación se emitirá la segunda línea (15), y el ordenador volverá a esperar su entrada. Escriba aquí la instrucción 'PRINT "linea 15"'. Después cierre la entrada con la tecla 'ESC'. Aparece el mensaje 'READY', señalando que la numeración automática de líneas está desactivada.

Ahora vuelva a entrar la instrucción 'AUTO 10,5', para conocer una particularidad de la misma. El ordenador reacciona del modo siguiente:

```
10*
```

¿Qué significa ahora el asterisco? Si ha leído con atención la descripción del comando, conocerá la respuesta correcta. El asterisco señala, que esta línea ya existe en el programa. Esto es importante para evitar, que se reescriba involuntariamente una misma línea, borrando su primera programación. Si usted desea, que dicha línea no sea modificada, pulse simplemente la tecla 'RETURN'. Si, por el contrario, desea modificarla, escriba la nueva línea y pulse 'RETURN'. Haga la prueba, conservando en principio la línea 10 con 'RETURN', y entrando en la línea 15 la instrucción 'PRINT "nueva linea 15"'. Después finalice el comando AUTO con la tecla 'ESC' y observe el resultado con ayuda del comando 'LIST'.

Usted acaba de conocer una posibilidad de simplifciar la entrada de un programa. Para adquirir mayor práctica, aplíquela también al efectuar las entradas de los ejemplos indicados en el presente libro.

RENUMERACION DE LINEAS

Ya sabemos, que la amplitud de paso entre las líneas de un programa no tiene ninguna importancia. La decisión de incrementar los números en 1, 5 ó 10 depende tan sólo de usted. Sin embargo tenga en cuenta, que una amplitud demasiado pequeña puede dificultar la inserción de líneas adicionales. Veamos un pequeño ejemplo práctico. Entre, en primer lugar, el siguiente programa:

```
1 CLS
2 PRINT "Vamos a multiplicar dos numeros:"
3 PRINT "El primer numero es 15"
4 PRINT "El segundo numero es 12"
5 PRINT "El resultado es";15*12
```

Como puede ver, este programa ha sido numerado de 1 en 1. El resultado del mismo es el siguiente:

```
Vamos a multiplicar dos numeros:
El primer numero es 15
El segundo numero es 12
El resultado es 180
```

Ahora queremos subrayar la primera línea para mejorar la emisión ópticamente. Por tanto, queremos que la emisión en pantalla muestre el siguiente aspecto:

```
Vamos a multiplicar dos numeros:
--------------------------------
El primer numero es 15
El segundo numero es 12
El resultado es 180
```

Para obtener este resultado, es necesario insertar una línea después de la línea 2 del programa. En este caso es necesario incrementar en 1 los números de las líneas siguientes a la 2, permitiendo así la entrada de una nueva línea llamada 3. El programa completado será:

```
1 CLS
2 PRINT "Vamos a multiplicar dos numeros:"
3 PRINT "--------------------------------"
4 PRINT "El primer numero es 15"
5 PRINT "El segundo numero es 12"
6 PRINT "El resultado es";15*12
```

¿Cómo resolvería ahora el problema? Seguramente volvería a entrar las líneas 3 a 5 con sus nuevos números. Pero ésta no es la solución. Su confortable BASIC dispone de un comando para solucionar este tipo de problemas, cuya descripción vemos a continuación:

Problema: **Renumeración de líneas**

Instrucción: **RENUM** nuevo, viejo, incremento

Parámetros: nuevo - **nueva línea inicial**
viejo - **vieja línea inicial**
incremento - **nueva amplitud de paso**

Ejemplo: **RENUM 100,10,15**
A partir de la línea 10 del programa, se renumera el programa empezando por la línea 100, y con un incremento de 5.

Observación: **La última línea del programa no puede superar el número 65535 tras la renumeración, de lo contrario no se ejecutará la instrucción RENUM y se emitirá el mensaje de error "Improper argument"**

Este comando es un poco más complicado que los explicados hasta ahora. Pero no se desanime. Como puede ver, hay que añadir tres parámetros a este comando. Se trata, en primer lugar, del número de la primera línea utilizada en la renumeración. Le sigue el número de la línea, a partir de la cual debe empezar la renumeración. Ya ve, que no es imprescindible renumerar el programa entero. Finalmente, el tercer parámetro indica la nueva amplitud de paso.

En nuestro ejemplo, ¿cuál debe ser la sintaxis de la instrucción RENUM para convertir las líneas 3 a 5 en 4 a 6? Vayamos paso a paso. En primer lugar, hay que indicar el nuevo número de la primera línea, o sea el 4. El primer número viejo representa el segundo parámetro, y es el 3. El incremento de paso continuará siendo 1. En definitva, la instrucción será:

```
RENUM 4,3,1
```

Esta instrucción conduce al resultado deseado, que podemos verificar mediante 'LIST':

```
1 CLS
2 PRINT "Vamos a multiplicar dos numeros:"
4 PRINT "El primer numero es 15"
5 PRINT "El segundo numero es 12"
6 PRINT "El resultado es";15*12
```

La instrucción anterior ha configurado el programa de tal modo, que ahora tenemos sitio para insertar la línea 3:

```
3 PRINT "---------------------------------"
```

Para evitar este tipo de problemas en el futuro, trabajaremos habitualmente con una amplitud de paso de 10. ¿Cuál debe ser la sintaxis de la instrucción 'RENUM', si queremos renumerarlo a partir de la línea 100, y con un incremento de 10? La solución no debería representarle ningún problema. Escriba la instrucción siguiente:

```
RENUM 100,1,10
```

El comando RENUM no sólo renumera las líneas, sino que además efectúa algo muy innportante: también cambia las instrucciones de salto a las líneas en cuestión. ¿Qué significa esto? Probémoslo. Añada otra línea al programa anterior:

```
100 CLS
110 PRINT "Vamos a multiplicar dos numeros:"
120 PRINT "---------------------------------"
130 PRINT "El primer numero es 15"
140 PRINT "El segundo numero es 12"
150 PRINT "El resultado es";15*12
160 GOTO 110
```

El último comando ha generado un bucle interminable. Si ahora renumeramos el programa de nuevo, la línea 110 recibe otro número. En este caso, la instrucción GOTO ya no debe saltar a la línea 110, porque ésta ya no existe una vez efectuada la renumeración. Para evitarlo, la instrucción RENUM no sólo cambia los números indicados al principio de cada línea, sino también las direcciones de salto hacia dichas líneas. Esto es sumamente importante, porque no podríamos utilizar el comando RENUM, si no efectuase dicha operación. Verifíquelo usted mismo, entrando la instrucción 'RENUM 10,100,5'. Si ahora visualiza el programa con 'LIST', observará, que la dirección de salto después del GOTO también ha sido modificada adecuadamente.

Ahora, usted ya domina el empleo de un comando, que le será de gran ayuda en el futuro.

BORRADO DE LINEAS DETERMINADAS

Ya conocemos el comando 'NEW', que borra un programa de la memoria. ¿Pero qué sucede, si no queremos borar el programa completo, sino únicamente algunas de sus líneas? El borrado de una única línea no representa ningún problema. Simplemente entramos el número de la línea en cuestión y, a continuación, pulsamos RETURN. Vamos a comprobarlo, borrando en primer lugar el programa anterior y entrando uno nuevo. Utilice el comando AUTO:

```
10 PRINT
20 PRINT
30 PRINT
40 PRINT
50 PRINT
60 PRINT
70 PRINT
80 PRINT
90 PRINT
```

Este programa no hace más que emitir nueve líneas en blanco, pero esto ya es suficiente para realizar nuestros ejercicios siguientes. Ahora borre la línea 50, entrando el número 50 seguido de la tecla 'RETURN'. Al listar el programa, verá el resultado de la operación: la línea 50 ha desaparecido. Sin embargo, ¿qué ocurre si deseamos borrar varias líneas, por ejemplo, de la 20 a la 40? Cierto, podemos borrar cada una de ellas, siguiendo el método explicado anteriormente, pero esto es bastante engorroso cuando se trata de borrar un número elevado de líneas. Existe otro comando, que nos facilita esta tarea:

Problema: **borrado de determinados grupos de líneas**

Instrucción: **DELETE** principio-final

Parámetros: principio - **primera línea a borrar**
final - **última línea a borrar**

Ejemplo: **DELETE 40-130**
borra las líneas 40 a 130

Ya tenemos la solución: para borrar las líneas 20 a 40 de nuesto programa, escribimos simplemente la instrucción:

DELETE 20-40

Disponemos de varias opciones para indicar los dos parámetros. Véalas en la siguiente tabla:

DELETE 10	**borra la línea 10**
DELETE 100-150	**borra las líneas 100 a 150**
DELETE 150-	**borra todas las líneas a partir de 150 incl.**
DELETE -100	**borra todas las líneas empezando por la 100**

La posibilidad de variar de tal forma el grupo de líneas, también se da para el comando LIST. El siguiente cuadro demuestra otra vez lo que queremos decir:

LIST 10	**lista la línea 10**
LIST 100-150	**lista las líneas 100 a 150**
LIST 150-	**lista todas las líneas a partir de la 150 incl.**
LIST -100	**lista todas las líneas empezando por la 100**

LAS TECLAS DE FUNCION

Ahora va a conocer una de las ayudas más confortables de la programación. A menudo es bastante molesto volver a teclear continuamente los comandos utilizados con mayor frecuencia. El CPC6128 le brinda la posibilidad de crear cadenas de caracteres utilizadas a menudo con la simple pulsación de una tecla. Para ello dispone de las 12 teclas de su teclado numérico:

7	8	9
4	5	6
1	2	3
0	.	ENTER

Puede asignar una cadena de caracteres de un máximo de 32 caracteres a cada una de estas teclas, pero procurando no superar un total de 120 caracteres. Veamos la descripción del correspondiente comando:

Problema: **Programar las teclas de función**

Instrucción: KEY n, string

Parámetros:

	n –	**número de la tecla de función**
	string –	**cadena de caracteres (max. 32)**

Ejemplo: **KEY 0,"LIST"**
Programa el comando LIST en tecla de función 0

Observación: **El número total de caracteres programados no debe superar 120.**

¿Verdad que es un comando sumamente interesante? Vamos a aplicarlo inmediatamente. El teclado numérico contiene 13 teclas de función, numeradas de 0 a 12. Seguramente se preguntará, cómo es posible obtener 13 teclas de función con tan sólo 12 teclas. Pues es posible debido a que la tecla 'ENTER' de dicho teclado tiene dos funciones: una activada por su pulsación normal, y otra pulsando simultáneamente la tecla 'CTRL'. El siguiente cuadro le muestra la numeración asignada a cada una de las teclas:

7	8	9
4	5	6
1	2	3
0	10	11/12

Por consiguiente, la tecla '.' tiene el número 10, y la tecla 'ENTER' los números 11 y 12. Vayamos a programar ahora estas teclas con algunos comandos prácticos. El comando 'LIST' p.ej. es utilizado con bastante frecuencia. Ejecuta la instrucción siguiente:

```
KEY 0,"LIST"
```

¡No se olvide nunca de poner un espacio en blanco entre el comando y el número! De lo contrario, su ordenador no se lo perdonará (Syntax error). Si ahora pulsa la tecla de función '0', aparecerá el comando 'LIST' en la pantalla. Sin embargo, este comando no será ejecutado hasta que haya pulsado 'RETURN'. Pero incluso el 'RETURN' puede provocarse con la misma tecla de función. Para ello debe añadirle el código interno de 'RETURN' con un comando especial que esxplicaremos más adelante. El código es 13, y la instrucción quedará como sigue:

```
KEY 0,"LIST"+CHR$(13)
```

Esta instrucción trae muchas novedades, que no deben confundirlo. En primer lugar, es importante conocer su efecto. Pulse la tecla de función '0' y observe, que el comando LIST es ejecutado inmediatamente.

Sin embargo, también existen comandos, que no deberían ir seguidos de un 'RETURN'. Uno de estos comandos es 'NEW'. Imagíne, que este comando está programado en un tecla de función, seguido del comando 'RETURN'. En ese caso, si pulsa involuntariamente esta tecla, se borrará el programa con el cual está trabajando. Por consiguiente, es aconsejable no añadir nunca 'RETURN' a aquellos comandos, que pueden causar efectos negativos al activarlos involuntariamente.

Ya conocemos el código de 'RETURN'. Relacionado con las teclas de función, existe otro código importante, el de las comillas. ¡Intente programar el comando 'RUN"' en la tecla de función 1! ¡Estoy seguro, de que le saldrán canas al intentarlo! ¿Cuál es el problema? El comando 'KEY 1,"RUN""' conduce a un 'Syntax error'. El ordenador interpreta el segundo signo de comillas como final de la cadena de caracteres, y no sabe qué hacer con el tercero. ¡Por consiguiente, en este caso debemos utilizar el código correspondiente a las comillas (34)! El comando correcto será:

```
KEY 1,"RUN"+CHR$(34)
```

Si ahora queremos añadirle además un 'RETURN', la instrucción tendrá una apariencia más complicada:

```
KEY 1,"RUN"+CHR$(34)+CHR$(13)
```

¡Simplemente hay que recordar los dos códigos 13 y 34!

Con el tiempo, usted habrá diseñado una programación estándar de sus teclas de función. No será necesario volver a programarlas después de cada encendido del ordenador. Puede memorizar el conjunto de comandos KEY en un programa, que puede grabar en el diskette, de forma que sea accesible en cualquier momento. Dicho programa podría ser, por ejemplo:

```
10 KEY 0,"LIST"+CHR$(13)
20 KEY 1,"PRINT "
30 KEY 2,"NEW"
40 KEY 3,"RENUM "
50 KEY 4,"AUTO "
60 KEY 5,"DELETE "
```

Confeccione usted mismo su propio programa de las teclas de función, mientras estudiemos el capítulo siguiente.

Otra observación para finalizar: ¡el mando de tres dedos (SHIFT/CTRL/ESC), que provoca el estado de conexión del ordenador, anula todas las teclas de función!

CAPITULO 5:

INTRODUCCION AL BASIC

El presente capítulo sirve de introducción al lenguaje de programación BASIC. Pero no queremos describir simplemente todos los comandos del BASIC uno tras otro, sino que iremos creando paso a paso un programa de gestión de direcciones, en el cual introduciremos y explicaremos cada comando en el momento oportuno. No pretendemos, que el lector de este libro se convierta en un perfecto programador con ayuda del presente capítulo, sino que pueda formarse una idea intensa de la programación práctica. Esta idea le servirá de base para continuar su formación con otros libros técnicos.

PROBLEMATICA DE LA GESTION DE DIRECCIONES

La gestión de direcciones es uno de los programas más apreciados para los ordenadores domésticos. Su amplia capacidad de aplicación contribuye en gran medida a este hecho. No existe prácticamente nadie que no sepa aplicar dicho programa. Es cierto que, cuando tenemos pocas direcciones, su gestión por ordenador no es necesariamente más efectiva que el método habitual de la pequeña agenda, pero es mucho más espectacular. Sin embargo, cuando se van acumulando muchas direcciones, el ordenador es el mejor lugar para guardarlas.

¿Qué posibilidades debe ofrecer un programa de este tipo? Bueno, en cualquier caso se trata de introducir y sacar direcciones.

Antes de pedirle más facultades al programa, debemos aclarar otro aspecto. En un programa de gestión de direcciones, se puede distinguir entre dos partes básicas:

1. El programa
2. Los datos

Los datos no forman parte del programa. Aunque fuese posible conseguirlo, en tal caso, los datos sólo podrían ser manejados por el programador, pero no por el usuario del programa. Cualquier modificación de los datos provocaría simultáneamente una modificación del programa, cosa que podemos descartar en seguida.

Por lo tanto, el programa se halla en un medio externo de almacenamiento (diskette), y será cargado en el ordenador cuando sea preciso. ¿Pero qué ocurre ahora con los datos? No tropezaremos con ellos hasta llegar al primer fichaje. Por tanto, ¿qué hacemos con ellos?

Se trata de organizar un fichero. Un fichero es un conjunto de datos, que se encuentra en un medio externo de almacenamiento. Aunque existen varias formas de organización de ficheros, aquí veremos únicamente el método más sencillo, el fichero secuencial. El término secuencial significa, que los datos están dispuestos uno tras otro. Los dos componentes más importantes de nuestra gestión de direcciones son:

1. El programa
2. El fichero

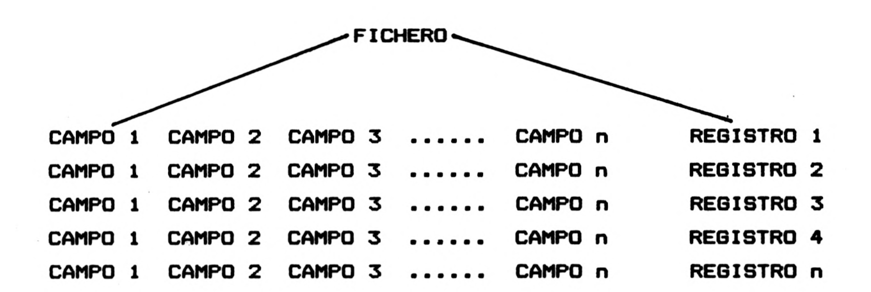
ORGANIZACION DE FICHEROS

Si yo preguntase ahora, cuál es la primera parte que debe organizarse, muchos contestarán: "el programa". Sin embargo, esta respuesta no es correcta. En primer lugar debemos saber exactamente, qué vamos a almacenar, cómo y dónde. Ya hemos resuelto el cómo, puesto que vamos a almacenar de forma secuencial.

Vamos a almacenar nuestros datos en la unidad de disco incorporada al ordenador.
Ahora sólo falta resolver la cuestión del qué es lo que vamos a almacenar. "Pues direcciones", contestarán muchos espontáneamente; pero todavía no nos hemos planteado, qué componentes deben formar parte de una dirección. Vamos a recoger todos los datos que pueden componer una dirección:

Nombre	**Código postal**
Apellido 1	**Población**
Apellido 2	**Teléfono**
Calle	**Observación**

Cada uno de estos componentes recibe el nombre de CAMPO. Así se habla, por ejemplo, del campo "nombre". El conjunto de los campos forma un REGISTRO. Por lo tanto, cada registro de nuestro ejemplo está formado por 8 campos. Finalmente, el conjunto de todos los registros forma el FICHERO. Observe esta estructura en el siguiente cuadro:



Por tanto, la jerarquía es FICHERO - REGISTRO - CAMPO.

MEMORIA INTERNA DE DATOS

Aunque el trabajo con un fichero secuencial es muy cómodo, también existen desventajas con respecto a otras formas de organización. Suponga que todo el fichero con sus direcciones se encuentra en un diskette. Entonces, usted desea localizar una dirección determinada del fichero. Aquí se presenta el problema. Es imposible leer un registro aislado de un fichero secuencial, la unidad de disco es incapaz de hacerlo. ¿Cómo podremos acceder, pues, a la dirección en cuestión?

Para acceder a un registro, hay que cargar todo el fichero en el ordenador. Por lo tanto, un fichero secuencial no debe ser mayor que la memoria disponible del ordenador. Sin embargo, esto también tiene una ventaja. Una vez cargado el fichero, el ordenador puede acceder con suma rapidez a los diversos registros. En consecuencia, el programa sigue el siguiente proceso:

1. Cargar fichero
2. Leer registro, modificar, borrar ...
3. Grabar fichero

Por lo tanto, después de iniciar el programa, en primer lugar debe leer el fichero completo de direcciones. Después podemos trabajar con cada uno de los ficheros. Pero antes de finalizar el programa, debemos volver a grabar el fichero, siempre y cuando lo hayamos modificado. En cambio, si los registros no han sido modificados, ni tampoco hemos añadido ni borrado nada, nuestro fichero actual coincide con el fichero que hemos cargado al principio. Por lo tanto, no es necesario volver a grabarlo.

VARIABLES

Ya sabemos, que el ordenador memoriza los registros. La pregunta es, ¿dónde y cómo los memoriza?

El ordenador memoriza los datos en forma de variables

Por tanto, 'variables' es la palabra mágica. Las variables son zonas de la memoria del CPC6128, provistas de un nombre. Al indicar dicho nombre, es posible acceder a estas zonas de la memoria. Ya conocemos dos tipos diferentes de datos. Existen los datos numéricos (números) y los alfanuméricos (Strings, texto). Las variables de cadena, o Strings, son identificadas mediante el carácter '$' detrás de su nombre de variable. ¿De qué se compone pues la designación de una variable? En primer lugar cabe mencionar, que el nombre de una variable no puede contener más de 40 caracteres. El primer carácter DEBE ser una letra, mientras que los demás pueden ser letras o números. Hay pocas excepciones. Como nombre de variable no pueden utilizarse ni comandos ni funciones del BASIC:

p.ej. PRINT, PI, KEY

Ejemplos de variables numéricas:

```
CONTADOR
CP
TELEFONO
CANTIDAD
```

Ejemplos de variables de cadena:

```
NOMBRE$
ARTICULO$
CUENTA$
X$
```

Es aconsejable elegir nombres lógicos para definir las variables. Así, por ejemplo, es conveniente designar la variable que memoriza el nombre con NOMBRE$. El precio de compra puede llamarse PRECIO$, etc. Los nombres lógicos de variables contribuyen considerablemente en una mejor comprensión y claridad del programa.

PROCESO DE VARIABLES

Volvamos ahora al ordenador. Empecemos a crear variables, a procesarlas y a emitirlas.

De hecho, el almacenamiento de datos numéricos en variables es un juego de niños. Supongamos, que se trata de almacenar el número 45.12 en la variable PRECIO. La correspondiente instrucción es:

```
PRECIO=45.12
```

¿Verdad que es sencillo? Si ahora queremos emitir dicha variable, la instrucción será:

```
PRINT PRECIO
```

Ahora entre ambas instrucciones en su ordenador. Para volver a borrar la variable PRECIO, simplemente debe dar la instrucción

```
PRECIO=0
```

También podemos utilizar las variables para efectuar operaciones matemáticas. Por ejemplo, vamos a emitir el valor duplicado de PRECIO. En el caso de que usted ya haya borrado la variable, vuelva a asignarle el valor anterior, escribiendo 'PRECIO=45.12':

```
PRINT PRECIO*2
```

El ordenador duplica el PRECIO y emite el resultado (90.24).

La propia variable mantiene su valor 45.12. ¿Pero qué debe hacerse para doblar el valor de la variable PRECIO? Puesto que se trata de asignar un nuevo valor a la variable PRECIO, la instrucción correspondiente empieza con PRECIO=', y se completará con la orden de doblar su valor:

```
PRECIO=PRECIO*2
```

Siempre se calcula primero el valor que se encuentra a la derecha del signo de igualdad, que, a continuación, es asignado a la variable que se encuentra a su izquierda.

Las variables de cadena o Strings se crean de forma parecida. Vamos a asignar el String "BARCELONA" a la variable de cadena CIUDAD$. La instrucción será la siguiente:

```
CIUDAD$="BARCELONA"
```

Para emitir las variables de cadena en la pantalla, también utilizamos el comando PRINT. Véalo usted mismo:

```
PRINT CIUDAD$
```

¿Qué sucede? Naturalmente se muestra el String almacenado en CIUDAD$. Si desea volver a borrar dicha variable, debe asignarle una cadena vacía.

```
CIUDAD$=""
```

Acabamos de ver, que una cadena vacía se compone de dos comillas seguidas de forma inmediata. Por supuesto, no podemos operar con el String, ni en el caso en que contenga un número. Esto significa, que, aunque usted entre p.ej. X$="123", no podrá efectuar ningún cálculo con X$.

Sin embargo, los Strings pueden concatenarse, tal como demuestra la siguiente secuencia de instrucciones:

```
CIUDAD1$="MADRID"
CIUDAD2$="BARCELONA"
CIUDAD$=CIUDAD1$+CIUDAD2$
PRINT CIUDAD$
```

En primer lugar, hemos almacenado las dos ciudades CIUDAD1$ y CIUDAD2$. A continuación hemos creado un nuevo String (CIUDAD$), que reúne los dos anteriores. Finalmente, la cuarta instrucción ha visualizado el String CIUDAD$.

Sería conveniente colocar un espacio en blanco entre ambas ciudades. Para ello debemos cambiar convenientemente la tercera instrucción:

```
CIUDAD$=CIUDAD1$+" "+CIUDAD2$
```

De esta forma hemos colocado un espacio en blanco entre los dos Strings.

Ahora ya conocemos los dos tipos más importantes de variables. La variable numérica y la de cadena o String, señalada con un '$' adicional. En este tipo de variables también pueden almacenarse direcciones, pero no es muy razonable. Para memorizar varios registros de estructura parecida en el ordenador, se precisan otro tipo de variables.

TABLAS

Las tablas tienen una aplicación frecuente en el proceso de datos. Si desea almacenar en el ordenador varios datos de un mismo grupo, es muy engorroso tener que asignar un nombre de variable a cada uno de sus elementos. En este caso, todos los nombres reciben el mismo nombre de variable, pero identificado con una señal adicional. Tomemos el ejemplo de un grupo de cinco ciudades:

MADRID

PARIS

LONDRES

ROMA

BONN

Para almacenar las cinco ciudades en el ordenador, precisaríamos cinco variables diferentes según el método anterior. Sin embargo, si almacenásemos este grupo de datos en una tabla (también llamada array), bastaría con un único nombre de variable. Sin embargo, puesto que se trata de diferenciar elementos de datos, deben ser señalados. Para ello se añade a la variable un número entre paréntesis. Este número se llama índice. Por lo tanto, el primer elemento recibe el índice 1, el segundo el índice 2, etc. En realidad, el índice empieza por 0 en lugar de 1, pero se pierde fácilmente la claridad, cuando el primer elemento recibe el número 0. Se puede ignorar el índice 0. De esta forma se pierde algo de memoria, pero al mismo tiempo se evita una posible fuente de errores.

Una variable de tablas o de array es, por ejemplo, DIRECCION$(1). XY(212) también es una variable numérica de array. Por lo tanto, se puede crear un índice de cualquier tamaño. Sólo debemos observar dos cosas:

1. **Si el índice supera el número 10, hay que reservar memoria para la tabla.**

2. **La dimensión de la tabla no puede superar la capacidad de la memoria.**

Por lo tanto, el ordenador procesa automáticamente tablas con un índice máximo de 10. ¿Pero cómo pueden definirse arrays de tamaño mayor? Para ello existe una comando específico del BASIC, el comando 'DIM'.

Problema: **Dimensionamiento de arrays**

Instrucción: **DIM** v1 (n1,n2,...)

Parámetros: v1 - **Nombre de variable del array**
n1,n2,...- **Indice máximo de cada dimensión**

Ejemplo: **DIM D$(20)**
Se trata de crear un array de nombre D$ hasta el índice 20.

Observación: **Cada array sólo puede ser dimensionado una vez en el programa, de lo contrario, se produce un "REDIM'D ARRAY ERROR".**

Cuando el índice máximo debe ser p. ej. 5, no se precisa ninguna instrucción DIM, puesto que el ordenador la procesa automáticamente hasta un índice de 10. Pero hay un pequeño inconveniente: si sólo se trabaja hasta un índice de 5, se reserva igualmente memoria para los índices 6-10, aunque ésta no sea ocupada. En este caso se puede evitar el problema, utilizando una instrucción DIM con el índice 5, con lo cual el ordenador sólo reservará la memoria necesaria para dicho índice.

Pero volvamos a nuestras cinco ciudades, que deben ser colocadas en una tabla. Si se utiliza la variable CIUDAD$, se almacenan las ciudades utilizando las siguientes asignaciones.

```
10 CIUDAD$(1)="MADRID"
20 CIUDAD$(2)="PARIS"
30 CIUDAD$(3)="LONDRES"
40 CIUDAD$(4)="ROMA"
50 CIUDAD$(5)="BONN"
```

En este caso, también podríamos entrar previamente la instrucción 'DIM CIUDAD$(5)', si no precisamos los índices 6-10.

Cuando se utiliza un índice, que supera la tabla reservada previamente, Se emite el mensaje de **"Subscript out of range"**. El índice también recibe el nombre de 'Subscript'.

En otro ejemplo, vamos a colocar además en una tabla los cinco países cuyas capitales son las ciudades antes almacenadas. La tabla recibe el nombre de PAIS$. Hay que efectuar las siguientes asignaciones:

```
60 PAIS$(1)="ESPAÑA"
70 PAIS$(2)="FRANCIA"
80 PAIS$(3)="INGLATERRA"
90 PAIS$(4)="ITALIA"
100 PAIS$(5)="RFA"
```

Ahora hemos creado dos tablas, cuyos índices están relacionados íntimamente. Esto significa, que CIUDAD$(1) contiene la capital de PAIS$(1). En general, podemos afirmar, que CIUDAD$(X) contiene la capital de PAIS$(X).

El atractivo especial de las tablas consiste en el hecho de que los índices pueden ser tanto números, como variables o cualquier otra expresión. Por lo tanto, se permite utilizar perfectamente una variable de array definida por 'DS$(X-2*I+13)'. La ventaja de esta indicación es evidente.

Las dos tablas, que acabamos de crear, se llaman unidimensionales, puesto que sólo poseen un índice. No obstante, las tablas pueden tener un número cualquiera de índices. Las dos tablas creadas anteriormente, también pueden ser sustituídas por una tabla bidimensional. En primer lugar, observe la siguiente secuencia de instrucciones:

```
10 DIM TABLA$(5,1)
20 TABLA$(1,0)="MADRID"
30 TABLA$(2,0)="PARIS"
40 TABLA$(3,0)="LONDRES"
50 TABLA$(4,0)="ROMA"
60 TABLA$(5,0)="BONN"
70 TABLA$(1,1)="ESPAÑA
```

```
80 TABLA$(2,1)="FRANCIA"
90 TABLA$(3,1)="INGLATERRA
100 TABLA$(4,1)="ITALIA
110 TABLA$(5,1)="RFA"
```

La instrucción DIM de la línea 10 no es necesaria, puesto que el ordenador efectúa automáticamente 'DIM D$(10,10)', pero con esta instrucción se reservarían 11*11 (121) posiciones de memoria. Sin embargo, nosotros sólo precisamos 10 posiciones de la tabla. Al trabajar con tablas de dos o más dimensiones, es recomendable definir su tamaño con la mayor precisión posible, para evitar un despilfarro de la memoria.

Ya disponemos de una tabla bidimensional, cuyo segundo índice identifica el grupo de datos (capital o país) y el primero, la posición de la tabla. Esta tabla también puede representarse como matriz:

	0	1
1	MADRID	ESPAÑA
2	PARIS	FRANCIA
3	LONDRES	INGLATERRA
4	ROMA	ITALIA
5	BONN	RFA

Volvamos ahora a la gestión de direcciones. Este tipo de tablas bidimensionales es ideal para albergar un fichero secuencial de direcciones. El primer índice numera los registros, el segundo índice numera los campos de cada registro. Puesto que nuestros registros constan de 8 campos, el valor máximo del segundo índice es 8. El primer índice, o sea la cantidad de direcciones, es opcional. En este caso, vamos a fijar un máximo de 200 direcciones. El nombre de la tabla será DIRECCION$. Por lo tanto, la instrucción DIM para

crear dicha tabla será

```
DIM D$(50,8)
```

Más adelante explicaremos la forma de administrarla.

ENTRADA DE DATOS A TRAVES DEL TECLADO

En los ejercicios precedentes sólo hemos procesado y emitido datos. Pero lo verdaderamente interesante empieza, cuando el programa recibe datos a través del teclado. El comando correspondiente se denomina INPUT. Sin embargo, solamente INPUT no tiene ningún efecto, sino que va seguido de dos parámetros: un string, que se emite antes del propio INPUT, y una variable, que debe almacenar los datos entrados. Importante: El comando INPUT no puede ser utilizado en el modo directo, sino únicamente dentro de un programa. Pero observe, en primer lugar, su descripción:

Problema: **Entrada de datos a través del teclado**

Instrucción: **INPUT** string;v1

Parámetros: string - **cualquier cadena de caracteres**
v1 - **variable, en la cual se almacena la entrada**

Ejemplo: **INPUT "NUMERO";X**
Se emite el string "NUMERO" y, a continuación, se lee un valor numérico del teclado, que será almacenado en la variable X.

Observación: **Este comando no puede ser utilizado en modo directo.**

Ahora entre el comando NEW, que borra cualquier programa anterior de la memoria. Las siguientes líneas BASIC demuestran la aplicación del comando INPUT:

```
10 PRINT "CALCULO DE CIRCULOS"
20 PRINT "-------------------"
30 INPUT "DIAMETRO          ";DIAM
40 PRINT "AREA DEL CIRCULO:"(DIAM/2)^2*PI
```

Ejecute el programa con el comando RUN. En la línea 30 se pide el diámetro del círculo a calcular. La entrada siempre debe cerrarse con RETURN. Detrás del string que sigue al INPUT, hay que indicar un punto y coma. Pero también se puede evitar el uso del string. El siguiente programa lo confirma:

```
10 PRINT "CALCULO DE CIRCULOS"
20 PRINT "-------------------"
30 INPUT DIAM
40 PRINT "AREA DEL CIRCULO:"(DIAM/2)^2*PI
```

Evidentemente, no es necesario entrar de nuevo todo el programa, sino simplemente cambiar la línea 30. Como puede ver, el string delante del INPUT tiene una misión puramente explicativa. Sin embargo, la documentación del INPUT también puede ser realizada por un PRINT en la línea precedente:

```
10 PRINT "CALCULO DE CIRCULOS"
20 PRINT "-------------------"
25 PRINT "DIAMETRO          ";
30 INPUT DIAM
40 PRINT "AREA DEL CIRCULO:"(DIAM/2)^2*PI
```

Entre además la línea 25 y vuelva a ejecutar el programa. Su ejecución es idéntica a la primera versión. Es importante colocar el punto y coma detrás del PRINT en la línea 25. Si no lo indica, el comando INPUT será efectuado en la línea inmediatamente inferior. Pruébelo.

El comando INPUT también permite leer strings a través del teclado. En primer lugar, veamos un ejemplo. Borre el programa anterior con NEW.

```
10 PRINT "COMO SE LLAMA USTED?"
20 INPUT NOMBRE$
30 PRINT "BUENOS DIAS ";NOMBRE$;", COMO ESTA USTED?"
```

Las líneas 10 y 20 también pueden escribirse en una sola línea. En este caso, el string delante del INPUT debe indicarse dentro de la instrucción INPUT.

```
10 INPUT "COMO SE LLAMA USTED?";NOMBRE$
20 PRINT "BUENOS DIAS ";NOMBRE$;", COMO ESTA USTED?"
```

Estos ejemplos demuestran todo lo que ofrece el comando INPUT. Observe, que el programa también continúa, si no se efectúa ninguna entrada, es decir, al pulsar simplemente RETURN. En ese caso, las variables toman el valor 0 ó "" (cadena vacía), si se trata de strings.

BUCLES

Permítanos empezar con un ejercicio, que más tarde nos llevará al control de bucles. Se trata de leer cinco nombres a través del teclado. Los nombres serán almacenados en el array NOMBRE$. Utilizando los comandos conocidos hasta ahora, confeccionaríamos el siguiente programa:

```
10 PRINT"ENTRADA DE LOS NOMBRES"
20 PRINT"----------------------"
30 INPUT"NOMBRE ";NOMBRE$(1)
40 INPUT"NOMBRE ";NOMBRE$(2)
50 INPUT"NOMBRE ";NOMBRE$(3)
60 INPUT"NOMBRE ";NOMBRE$(4)
70 INPUT"NOMBRE ";NOMBRE$(5)
```

Este programa leerá ahora los cinco nombres, uno tras otro. ¿Pero no es un procedimiento muy complicado? ¡Imagínese, si se tratase de leer 100 nombres!

En la práctica, no verá nunca semejante estilo de programa. Para este fin se utilizan los bucles. ¿Pero cómo se construye un bucle? En primer lugar, vea la descripción del comando correspondiente:

Problema: **Control de bucles**

Instrucción: **FOR** v1 = n1 **TO** n2 **STEP** n3

Parámetros: v1 - **variable numérica**
n1 - **valor inicial de** v1
n2 - **valor final de** v1
n3 - **incremento, valor en que se incrementa** v1 **tras cada ejecución del bucle.**

Ejemplo: **FOR A=1 TO 20 STEP 1**
La variable A se incrementa en 1 después de cada pasada, hasta alcanzar el valor 20 (20 ejecuciones del bucle)

Observación: **Si el incremento es 1, se puede omitir el parámetro STEP. Por lo tanto, la instrucción del ejemplo anterior también podría escribirse:**
FOR A=1 TO 20

Cada bucle empieza con una instrucción FOR. Todos los comandos posteriores pertenecerán al bucle y, por lo tanto, serán ejecutados varias veces. ¿Pero cómo se señala el final del bucle? Para ello disponemos de otra instrucción, cuya misión consiste en comprobar, si la variable del bucle ha alcanzado el valor final. Si todavía no lo ha alcanzado, se incrementa la variable del bucle y se vuelve a ejecutar el mismo. Sin embargo, cuando la variable haya alcanzado su valor final, la ejecución del programa continuará después del comando NEXT.

Problema: **Definir final del bucle**

Instrucción: **NEXT** v1

Parámetros: v1 - **variable del bucle**

Ejemplo: **NEXT A**

Señala el final del bucle contenido en el ejemplo anterior.

Observación: **Si no se indica la variable** v1, **el comando NEXT hace referencia al último bucle creado.**

Ahora podemos programar cualquier bucle con ayuda de estos dos comandos. Volvamos de nuevo a la anteriormente descrita entrada de datos mediante INPUT. Se trataba de introducir cinco nombres en un array con los índices 1 a 5. Si ahora utilizamos un bucle, precisaremos sólo una instrucción INPUT, en la cual se aplica el índice como variable, la variable del bucle. El bucle debe ejecutarse de 1 a 5. ¿Cómo debe escribirse la correspondiente instrucción FOR con todos sus parámetros?

```
FOR I=1 TO 5
```

El programa completo modificado es el siguiente:

```
10 PRINT "ENTRADA DE LOS NOMBRES"
20 PRINT "----------------------"
30 FOR I=1 TO 5
40 INPUT "NOMBRE ";NOMBRE$(I)
50 NEXT I
```

¿Verdad que es una solución muy elegante para leer 5 strings? Veamos los pasos seguidos por el programa tras iniciarlo. En primer lugar, se ejecutan las dos instrucciones PRINT. Después comienza el bucle con la variable I, el valor inicial 1, el valor final 5 y el incremento 1. Recuerde: si no indicamos ningún parámetro STEP, el programa trabaja con el incremento 1. Al iniciar el bucle, I obtiene el valor 1. Por

lo tanto, en la línea 40 se lee la variable del array cuyo índice es 1. A continuación, el comando NEXT comprueba, si se ha alcanzado el final del bucle (variable I mayor que 5). Puesto que no es así, se aumenta I en el incremento 1, y el programa prosigue su ejecución después del comando FOR, es decir, en la línea 40. La variable I obtiene ahora el valor 2. El siguiente INPUT transfiere el próximo nombre al array, con el índice 2. Y así sucesivamente, hasta que el comando NEXT dé por terminado el bucle. Cuando al valor de la variable sea 4, volverá a incrementarse en 1 y se volverá a ejecutar el bucle, siendo el valor I=5. A continuación, el comando NEXT comprueba, si la variable I ha alcanzado su valor final (5). En caso afirmativo, el programa prosigue, o sea, termina su ejecución después de la instrucción NEXT.

Ahora podemos ampliar el programa, haciendo que vuelva a mostrar todos los nombres después de haberlos entrado. Para ello es necesario añadir otro bucle:

```
60 FOR I=1 TO 5
70 PRINT "NOMBRE ";NOMBRE$(I)
80 NEXT I
```

Por lo tanto, no es todo tan complicado como puede haber pensado en principio.

¿Qué otras aplicaciones pueden tener los bucles? Por ejemplo, podemos crear un bucle de espera para demorar la ejecución del programa. Para ello sólo debe crear un bucle con FOR y cerrarlo inmediatamente después con NEXT. Es importante saber, cuántas veces debe ejecutarse un bucle de este tipo para crear, por ejemplo, un bucle de espera de 1 segundo. Se puede partir del principio, de que un bucle de estas características se ejecuta 1000 veces por segundo. Por lo tanto, un bucle de espera de unos 3 segundos será:

```
FOR X=1 TO 3000:NEXT
```

Recuerde: podemos escribir varias instrucciones en una sola línea, si las separamos entre sí mediante el signo de dos puntos. Ahora vamos a introducir un bucle de espera en nuestro programa, situado entre la entrada y la emisión de los datos. Un número de línea, que se encuentre entre ambos sucesos, puede ser p. ej. la línea 55. ¿Cómo debemos escribir dicha línea, si deseamos retardar el programa unos 2 segundos?

```
55 FOR X=1 TO 2000:NEXT
```

Ahora visualice con LIST el programa entero en la pantalla. Verá, que todas las líneas escritas posteriormente han sido ordenadas adecuadamente. Si ahora ejecuta el programa con RUN, podrá observar el efecto producido por la línea 55.

Hasta ahora sólo hemos trabajado con un incremento de 1, que también puede tomar cualquier otro valor. Si deseamos leer los nombres del ejemplo anterior de 5 a 1, en lugar de hacerlo de 1 a 5, también podemos conseguirlo fácilmente. En ese caso, el valor inicial del bucle será 5, su valor final 1 y el incremento, es decir, el parámetro STEP será -1. ¿Cómo habrá que modificar las líneas 30 y 60? Seguramente ya dispondrá de la solución adecuada:

```
30 FOR I=5 TO 1 STEP -1
40 FOR I=5 TO 1 STEP -1
```

Hagamos más ejercicios sobre el comando FOR. Se trata de construir los siguientes bucles con ayuda del comando FOR:

	VARIABLE DE BUCLE	VALOR INICIAL	VALOR FINAL	INCREMENTO
A)	INDICE	10	18	1
B)	NUMERO	30	15.5	-0.5
C)	BUCLE	590	1800	0.25
D)	TIEMPO	1	10	0.3

Las soluciones:

A) FOR INDICE=10 TO 18 STEP 1
o bien
FOR INDICE=10 TO 18
B) FOR NUMERO=30 TO 15.5 STEP -0.5
C) FOR BUCLE=590 TO 1800 STEP 0.25
D) FOR TIEMPO=1 TO 10 STEP 0.3

Otra observación para finalizar. Si el bucle creado con el comando FOR es lógicamente incorrecto (por ejemplo, FOR I=10 TO 0), el bucle siempre se ejecutará una vez.

PRIMERAS REACCIONES DEL PROGRAMA

Después de avanzar hasta aquí en la programación BASIC, ya es hora de empezar con el programa de direcciones. ¿Cuál será su principio más adecuado? Si usted ya ha manejado programas similares, seguramente sabrá lo que un programa de este tipo muestra en la pantalla después de iniciarlo. Suele anunciarse con un encabezamiento del programa. En primer lugar, vamos o crear este encabezamiento, compuesto por un número reducido de instrucciones PRINT. No se deje intimidar por las instrucciones completamente nuevas para usted y entre el principio del programa, ejecutándolo a continuación. Después le explicaremos dichas instrucciones.

```
100 REM ================================
110 REM   Encabezamiento del programa
120 REM ================================
130 CLS
140 PRINT STRING$(40,"=")
150 LOCATE 12,2
160 PRINT "GESTION DE DIRECCIONES"
170 PRINT STRING$(40,"=")
```

Después de entrar y ejecutar esta parte del programa, querrá conocer el significado de los nuevos comandos. Empecemos por las líneas 100-120. En ellas se documenta esta sección del programa. Para evitar, que el ordenador interprete la parte explicativa como instrucciones, dichas líneas se señalan con REM. REM significa REMARKS, es decir notas. Las líneas REM pueden añadirse en cualquier lugar del programa. La señalización de las diversas partes de un programa contribuye notablemente en su claridad.

El siguiente comando nuevo no es ningún comando en el sentido habitual, sino más bien una función, mejor dicho una función de String. Ya sabemos, que una variable de cadena recibe el nombre de String en el lenguaje especializado. Con ayuda de la función STRING$ puede emitirse un carácter repetido hasta un máximo de 255 veces. Veamos primero la descripción:

Problema: **Creación de cadenas de caracteres**

Instrucción: **STRING$(n,"z")**

Parámetros: n - **Cantidad de caracteres (0-255)**
"z" - **Carácter a multiplicar**

Ejemplo: **PRINT STRING$(80,":")**
Se emiten 80 signos de dos puntos

Pruebe esta función en relación con el comando PRINT. Dada su simplicidad, vamos a explicar seguidamente el comando siguiente.

El comando PRINT emite datos por la pantalla. El lugar, en el cual se emitirán, depende del último comando PRINT o CLS. Si queremos obtener una emisión con PRINT en un lugar determinado de la pantalla, su BASIC dispone del siguiente comando:

Problema: **Posicionar el Cursor**

Instrucción: **LOCATE** c,l

Parámetros: c - **Columna**
l - **Línea**

Ejemplo: **LOCATE 5,10**
coloca el Cursor en la columna 5 de la línea 10

Después de cada instrucción PRINT, se modifica la posición del Cursor. Sin embargo, no podremos ver el Cursor en la pantalla, puesto que la emisión está controlada por el programa. Por consiguiente, si nosotros mismos queremos definir la posición, en la cual deberá efectuarse la emisión del comando PRINT, éste debe ir precedido del comando LOCATE. Tal como acabamos de explicar, con ayuda de LOCATE podemos indicar la columna y la línea que definen la posición. Veamos un ejemplo para entendernos mejor. Pretendemos, que el texto '**HOLA**' se muestre en el centro de la pantalla, utilizando para ello los comandos PRINT y LOCATE. La instrucción será la siguiente:

```
LOCATE 18,13:PRINT "HOLA"
```

Recuerde: podemos escribir varias instrucciones en una sola línea, siempre y cuando estén separadas por dos puntos.

Pero volvamos ahora a nuestro programa.

SUBPROGRAMAS

En el siguiente transcurso de la gestión de direcciones, continuaremos utilizando con frecuencia las líneas BASIC, que editan el encabezamiento del programa. Para evitar teclearlas una y otra vez, lo cual también alargaría inútilmente la extensión del programa, vamos a convertirlas en un subprograma (también llamado rutina). ¿Pero cómo hay que señalar un subprograma y cómo ejecutarlo? Dos preguntas, que contestaremos inmediatamente, como de costumbre.

En el capítulo anterior ya hemos explicado el comando GOTO, que permite continuar el programa en cualquier línea. Sin embargo, puesto que el programa debe volver al punto de partida después de ejecutar un subprograma, y la rutina no puede conocer el lugar, donde ha sido llamada con GOTO, el comando GOTO no sirve para este propósito. Existe un comando parecido para llamar subrutinas, que describimos a continuación.

Problema: **Llamada de subprogramas (rutinas)**

Instrucción: **GOSUB** nl

Parámetros: nl - **número de línea, a partir de la cual empieza la rutina.**

Ejemplo: **GOSUB 100**
llama un subprograma en la línea 100

Observación: **- Los subprogramas llamados con GOSUB deben ser cerrados con el comando RETURN.**
- Si, en su ejecución, el programa encuentra un RETURN, sin haber efectuado previamente un comando GOSUB, se emite el mensaje de error "Unexpected RETURN"

Esto parece ser muy sencillo, y lo es, después de aplicarlo unas cuantas veces. Ahora vamos a convertir las líneas

100-170 en un subprograma, que pueda ser llamado desde cualquier lugar en un programa posterior. Tal como habrá observado en la descripción del comando, debemos cerrar el subprograma con el comando RETURN. Para ello le añadimos una línea:

```
180 RETURN
```

Sin embargo, ahora ya no podemos ejecutar este programa. Si lo ejecuta, aparece el mensaje de error **"Unexpected RETURN in 180"**. Esto es lógico, porque accedemos con RUN a un subprograma, que únicamente puede ser llamado con GOSUB. ¿Cómo continuamos ahora?

En primer lugar, observe cómo se organizan los números de línea del futuro programa.

10-99 **Aquí se coloca toda la información previa del programa. Por ejemplo, creación de tablas, asignación de variables.**

100-999 **En estas líneas se colocan todas las rutinas.**

1000- **A partir de la línea 1000 empieza el programa propiamente dicho, el programa principal.**

Por lo tanto, debemos saltarnos el subprograma para llegar al programa principal. Para ello, entramos la línea 99:

```
99 GOTO 1000
```

En la línea 1000 empieza el programa principal, hecho que señalaremos primero con líneas REM, añadiendo después el comando GOSUB:

```
1000 REM =======================
1010 REM    PROGRAMA PRINCIPAL
1020 REM =======================
1030 GOSUB 100
```

Ahora, el programa vuelve a ser ejecutable. Al principio se ejecuta el subprograma, que visualiza el encabezamiento del programa.

EL MENU

En primer lugar, veamos los progresos de nuestro programa:

```
99 GOTO 1000
100 REM ===============================
110 REM   ENCABEZAMIENTO DEL PROGRAMA
120 REM ===============================
130 CLS
140 PRINT STRING$(40,"=")
150 LOCATE 12,2
160 PRINT "GESTION DE DIRECCIONES"
170 PRINT STRING$(40,"=")
180 RETURN
1000 REM =========================
1010 REM    PROGRAMA PRINCIPAL
1020 REM =========================
1030 GOSUB 100
```

Ahora, el programa se presentará por su nombre al iniciar su ejecución. Pero seguramente esto no es todo, sino que no es más que un pequeño principio. Puesto que deseamos, que este programa cuente con varias posibilidades de procesar las direcciones, es necesario ofrecerlas. Reflexionemos, qué facultades debe tener nuestro programa. En primer lugar, queremos que sea capaz de grabar los datos en un medio externo de almacenamiento, así como de volver a cargarlos del mismo. Para ello, precisamos de las dos opciones siguientes:

-1- CARGAR FICHERO
-2- GRABAR FICHERO

Además, es preciso entrar direcciones. Para ello utilizamos la función 3:

-3- ENTRAR DIRECCIONES

También debe ser posible modificar los datos entrados. Ello se consigue con el punto 4:

-4- MODIFICAR DIRECCIONES

Si no queremos saber nada más del tío Carlos a partir de este mismo instante, lo mejor será borrarlo del fichero de direcciones. Por lo tanto, necesitamos una función de "borrado":

-5- BORRAR DIRECCIONES

Ahora llega lo más importante: todos los datos nos sirven poco, si no podemos emitirlos o llamarlos. El siguiente programa parcial sirve para emitir direcciones:

-6- EMITIR DIRECCIONES

No es recomendable, que ningún programa sea finalizado con el interruptor de corriente del ordenador. Por esta razón, el último punto crea una salida del programa:

-7- FINALIZAR PROGRAMA

Una vez conocidas todas las funciones del programa, vamos a ampliarlo de tal forma, que dichas funciones sean visualizadas en forma de menú, añadiéndole las siguientes instrucciones:

```
1040 LOCATE 1,7
1050 PRINT "      FUNCIONES DEL PROGRAMA:"
1060 PRINT "      ------------------------"
1070 PRINT
1080 PRINT "      -1- Cargar fichero"
1090 PRINT "      -2- Grabar fichero"
1100 PRINT "      -3- Entrar direcciones"
```

```
1110 PRINT "      -4- Modificar direcciones"
1120 PRINT "      -5- Borrar direcciones"
1130 PRINT "      -6- Emitir direcciones"
1140 PRINT "      -7- Finalizar programa"
```

Si ahora vuelve a ejecutar el programa observará, que ya empieza a adquirir un carácter profesional. El usuario debe decidir ahora, cuál de las funciones precisa para trabajar. Con este fin, leemos una variable numérica, que debe contener el valor 1 a 7, según la función escogida. Pero antes emitimos otro PRINT, que contiene **"ELECCION"**.

```
1150 LOCATE 10,18:PRINT "Elección ";
1160 INPUT funcion
```

LA CONSULTA CON IF

¿Pero qué ocurre, si el usuario entra un número inválido, como por ejemplo 9? Es aconsejable, que el programa lo impida. Sólo hace falta saber cómo hacerlo. Debemos averiguar, qué valor ha sido tecleado, es decir, debemos comprobar, si el valor de la variable 'F' no es válido. Vamos a describir la instrucción correspondiente:

Problema:	**Comprobar una condición**	
Instrucción:	**IF** condición **THEN** instrucción	
Parámetros:	condición	- **operación de comparación (p.ej. A=10 o B>12)**
	instrucción	- **comandos, que serán ejecutados, si se cumple la condición.**
Ejemplo:	**IF F=0 THEN GOTO 1000** **Si el valor de la variable F es 0, el programa se bifurca hacia la línea 1000; de lo contrario prosigue su ejecución en la línea siguiente.**	

Observación: **Si THEN va seguido de GOTO, este último puede ser ignorado. El ejemplo anterior también podría escribirse así:**
IF F=0 THEN 1000

Una condición no sólo puede contener el operador de comparación '=', sino también los operadores '>' (mayor que) y '<' (menor que). También se admiten combinaciones de los tres operadores. En resumen, podemos utilizar los siguientes operadores de comparación:

Símbolo	**Significado**
=	**igual a**
>	**mayor que**
<	**menor que**
>=, =>	**mayor o igual que**
<=, =<	**menor o igual que**
<>, ><	**distinto de**

Por lo tanto, el comando IF comprueba en primer lugar, si se cumple la condición. En caso afirmativo, se ejecutarán las instrucciones que siguen al comando THEN. Si la condición no se cumple, el programa continúa su ejecución después de la línea IF.

Pero esto no es todo lo que puede ofrecer una lectura IF. También permite combinar varias condiciones lógicas entre sí. Esto parece más complicado de lo que es en realidad. Veámoslo con ayuda de un ejemplo:

```
IF RESP=1 OR RESP>12 THEN 1000
```

Dos instrucciones combinadas entre sí. Sabiendo que OR significa O, podemos interpretar la instrucción de la forma siguiente: si RESP es igual a 1 o RESP es mayor que 12, vé hacia 1000. Por lo tanto, para conseguir que se efectúe el salto hacia la línea 1000, debe cumplirse que RESP=1 o que RESP>12, o ambas condicones al mismo tiempo. Tan sólo si no

se cumple ninguna de las dos condiciones, la ejecución del programa continúa después de la línea IF. La combinación OR es una de las dos combinaciones posibles. Veamos otro ejemplo:

```
IF RESP=1 AND TEST>12 THEN 1000
```

En este caso se trata de una operación AND, que se cumple únicamente si RESP es igual a 1 y TEST mayor que 12.

Volvamos ahora a nuestro problema. Queríamos comprobar, si el usuario ha entrado un número, que no está comprendido entre 1 y 7. ¿Cómo debe codificarse la correspondiente instrucción IF? Vea la solución:

```
IF funcion=0 OR funcion>7 THEN ...
```

Bien, ¿qué sucederá? Emitiremos simplemente un mensaje de error en la última línea de la pantalla y volveremos a la línea 1000. Para emitir el mensaje de error, necesitaremos otro subprograma, que lea el tipo de error en un string y lo visualice en la línea 24. Analicemos el correspondiente subprograma:

```
200 REM =====================
210 REM    MENSAJE DE ERROR
220 REM =====================
230 LOCATE 1,24
240 PRINT error$
250 PRINT CHR$(7);
260 FOR I=1 TO 1000:NEXT I
270 LOCATE 1,24
280 PRINT STRING$(39," ")
260 RETURN
```

En primer lugar, la línea 230 posiciona el Cursor en la última línea. La línea 240 emite el String 'error$', activado por el programa que efectuó la llamada. Por consiguiente, antes de que esta rutina pueda ser llamada por el programa principal, debemos almacenar el mensaje de error a emitir en la variable de cadena 'error$'.

CODIGO ASCII

Ahora veamos la función de la línea 250, que ya utilizamos al trabajar con las teclas de función. Recuerde, que añadimos un RETURN -CHR$(13)- a las instrucciones para conseguir, que fuesen ejecutados automáticamente. Pero veamos ahora estas funciones con más detalle. El ordenador asigna un código a cada carácter, cuyo valor está comprendido entre 0 y 255. Esta codificación está normalizada según la norma ASCII americana. Los caracteres del CPC6128 difieren ligeramente de estos códigos. Pero no sólo se codifican los caracteres. También algunas funciones específicas, tales como la emisión de un sonido de nuestro ejemplo, pueden ser reclamadas a través de un código ASCII. A continuación describiremos dos funciones, que sirven para codificar caracteres según la norma ASCII:

Problema: Convertir un código ASCII en un carácter

Función: CHR$(n)

Parámetros: n - código ASCII (0-255)

Ejemplo: FOR I=32 TO 255: PRINT CHR$(I):NEXT I
visualiza todos los caracteres, que pueden representarse con el CPC6128

Observación: Cuando se indica un código ASCII superior a 255, se emite el mensaje "Improper argument"

Problema: Convertir un carácter en el código ASCII

Función: ASC("z")

Parámetros: "z" - un carácter, o una variable de cadena que contiene un carácter

Ejemplo: **PRINT ASC("A")**
visualiza el código ASCII de la letra "A" (65) en la pantalla.

Observación: **- Si la variable es una cadena vacía, se emite el mensaje de error "Improper argument"**
- Si la variable de cadena supera la extensión de un carácter, se buscará el código ASCII del primer carácter.

Ahora debemos digerir este montón de informaciones nuevas. Como ya hemos dicho, en el interior del ordenador sólo se trabaja con los códigos ASCII de los caracteres. Por lo tanto, el ordenador también nos entiende si, en lugar del propio carácter, entramos su código ASCII.

Si desea saber, cuál es el código ASCII de un carácter determinado, puede hallarlo con ayuda de la función ASC("z"). Para ello, simplemente debe escribir la instrucción 'PRINT ASC("z")' -sustituyendo z por el carácter en cuestión- en modo directo. El CPC6128 contestará inmediatamente con el correspondiente código.

En la línea 250 se genera pues un sonido con la instrucción 'PRINT CHR$(7)', que debe avisarnos acústicamente de que hemos cometido un error. Por cierto, este código también está codificado según la norma ASCII, y puede encontrarse en prácticamente todos los ordenadores.

Pero volvamos a nuestra rutina de error. La señal acústica va seguida de un bucle de espera, que genera una demora de aproximadamente 1 segundo para permitir la lectura del mensaje. A continuación, se vuelve a borrar el mensaje de error en la línea 280, escribiendo simplemente 39 caracteres en blanco encima de la línea. Finalmente, en la línea 290 se efectúa el salto de vuelta al programa principal.

Ahora podemos completar nuestra consulta con IF después del INPUT:

```
1170 If funcion=0 OR funcion>7 THEN error$=
"valor inválido":GOSUB 200:GOTO 1150
```

Ahora entre esta línea. Observe, que la repartición en dos líneas de texto no corresponde a la repartición en su pantalla. Si el usuario del programa indica ahora un 0 o un número mayor que 7, habrá cometido un error en su manejo, el cual le señalamos en la rutina 200. Para ello, en primer lugar, almacenamos el mensaje de error en el String, después llamamos la rutina de error, y finalmente bifurcamos hacia el nuevo INPUT.

Ahora ejecute el programa y entre un valor inválido. El programa detectará la entrada errónea y emitirá el mensaje de error.

GOTO CONDICIONADO

Ahora debemos iniciar los pasos necesarios, si el valor entrado ha sido correcto. Es decir, hay que bifurcar hacia la parte adecuada del programa.

Para efectuar un salto dentro del programa, que dependa de la variable **'funcion'**, disponemos de un comando que permite realizarlo del modo más sencillo. Vea la descripción de este comando:

Problema: Salto condicionado

Instrucción: **ON** v1 **GOTO** n1,n2,n3,...

Parámetros:

v1	-	**variable numérica**
n1,n2,n3,...	-	**números de línea, hacia las cuales debe dirigirse el salto, según el contenido de F**

Ejemplo: ON F GOTO 100,200,300,400
Si F=1, se bifurca hacia la línea 100, si F=2, se bifurca hacia la línea 200, etc. Si F es mayor que 4, no se realiza ningún salto, sino que la ejecución continúa en la línea siguiente

Observación: Si F no es un número entero (p.ej. 3.45), se utilizará la parte entera de dicho número.

Por lo tanto, es un comando ideal para nuestro objetivo. En primer lugar, definiremos los números de línea, donde deben comenzar las respectivas partes del programa. El comando siguiente debe escribirse de la forma:

```
1140 ON funcion GOTO 5000,10000,15000,
20000,25000,30000,35000
```

A continuación veremos, dónde se encuentra cada subprograma:

'función'	Línea	Subprograma
1	5000	CARGAR FICHERO
2	10000	GRABAR FICHERO
3	15000	ENTRAR DIRECCIONES
4	20000	MODIFICAR DIRECCIONES
5	25000	BORRAR DIRECCIONES
6	30000	EMITIR DIRECCIONES
7	35000	FINALIZAR PROGRAMA

A continuación iremos confeccionando un subprograma tras otro. Lógicamente, empezaremos con la función 3, ENTRAR DIRECCIONES. Pero antes hay que dimensionar la tabla que debe albergar las direcciones. Para ello añadimos la línea 10 a nuestro programa:

```
10 DIM direcciones$(200,7):REM Tabla de direcciones
```

Hemos reservado memoria para 200 direcciones. En general, esto es del todo suficiente. ¿Quién tiene más de 200 conocidos? Sin embargo, en el ámbito comercial, la aplicación del programa es limitada. Una empresa, que desee llevar la gestión de sus 2000 clientes, debería disponer de un programa más apropiado para ello.

Permítanos escribir delante de cada subprograma una rutina, que muestre un bonito encabezamiento para cada uno de ellos. Esta rutina también debe emitir previamente el encabezamiento del programa. Cada subprograma tiene un nombre, que debería visualizarse en el menú. Depositaremos estos nombres en una tabla, cuyo índice corresponda a la variable 'funcion'. ¿Qué significa esto? En el resumen de la última página verá, que el subprograma 'BUSCAR DIRECCIONES', por ejemplo, lleva el número 3. Este número indicará pues la posición dentro de la tabla. Asignaremos el nombre de **'funcion$'** a la tabla. De esta forma, la variable **'funcion$(3)'** p.ej. puede recibir el texto 'ENTRAR DIRECCIONES'. La creación de esta tabla forma parte de las preparaciones, por lo cual será colocada entre las líneas 0-99. Las siguientes líneas crean la tabla:

```
20 funcion$(1)="   Cargar fichero    "
21 funcion$(2)="   Grabar fichero    "
22 funcion$(3)=" Entrar direcciones  "
23 funcion$(4)="Modificar direcciones"
24 funcion$(5)=" Borrar direcciones  "
25 funcion$(6)=" Emitir direcciones  "
26 funcion$(7)=" Finalizar programa  "
```

Tenga en cuenta, que todos los Strings tienen la misma longitud (las comillas coinciden una debajo de otra). Ahora, a partir de la línea 300 del subprograma, queremos visualizar el encabezamiento del programa y, a continuación, la función seleccionada por el usuario:

```
300 REM ==============================
310 REM ENCABEZAMIENTO DE SUBPROGRAMAS
320 REM ==============================
330 GOSUB 100
```

```
340 LOCATE 8,5:PRINT STRING$(23,"*")
350 LOCATE 8,6:PRINT "*"+funcion$(funcion)+"*"
360 LOCATE 8,7:PRINT STRING$(23,"*")
370 RETURN
```

Aquí puede ver, que incluso un subprograma puede bifurcarse hacia otro subprograma. En primer lugar se muestra el encabezamiento general del programa. Después se crea la primera línea del cuadro que debe enmarcar el nombre de la función. La siguiente línea tiene una apariencia más complicada de lo que es en realidad. En primer lugar, el Cursor es posicionado en la columna 10 de la línea 6, donde se emite un asterisco seguido del nombre de la función, que procede de la tabla 'funcion'. Después se añade otro asterisco al String. La línea 360 crea la línea inferior del recuadro.

¿Pero cuándo accederemos a esta rutina? Lógicamente no podrá ser ejecutada, hasta que el usuario haya seleccionado una de las funciones, cuyo número esté contenido en la variable 'funcion'. Esto significa, que será después de la consulta con IF. Añada ahora la línea 1175:

```
1175 GOSUB 300
```

Hemos escogido esta línea, porque después de la línea 1180 se produce un salto hacia otro subprograma.

ENTRAR DIRECCIONES

Ahora la cosa empieza a ponerse seria. Hay que confeccionar el primer subprograma importante. Pero vayamos por partes, y empecemos por lo más sencillo:

```
15000 REM =======================
15010 REM    ENTRAR DIRECCIONES
15020 REM =======================
```

Hasta aquí no hay problema, pero ahora viene lo complicado (no se desanime). Debemos memorizar de algún modo la cantidad de registros que han sido entrados. Para ello, y en el presente subprograma, iremos incrementando un indicador que contiene el número de direcciones entradas. Le llamaremos simplemente **'indicador'**. La línea siguiente se encargará de incrementarlo de uno en uno.

```
15030 indicador=indicador+1
```

Por ejemplo, si **'indicador'** es distinto de 0, ya se habían cargado o entrado datos previamente, que ahora serán ampliados. En ese caso añadiremos automáticamente más direcciones al fichero.

Ahora sería conveniente entrar 7 líneas INPUT, que permitan leer los 7 campos de datos de una dirección. Pero esto es más sencillo. En primer lugar, almacenaremos todos los nombres de campo en el array (tabla) **'campo$(1)'** a **'campo$(7)'**, empezando por la línea 30.

```
30 campo$(1)="NOMBRE         "
31 campo$(2)="APELLIDO 1     "
32 campo$(3)="APELLIDO 2     "
33 campo$(4)="CALLE          "
34 campo$(5)="CP/POBLACION   "
35 campo$(6)="TELEFONO       "
36 campo$(7)="OBSERVACION    "
```

Estos nombres de campo se almacenan en el array al principio del programa, y pueden utilizarse en cualquier momento de su ejecución. En las siguientes líneas del programa observará el fin, con que serán utilizados ahora.

```
15040 PRINT
15050 FOR indice=1 TO 7
15060 PRINT campo$(indice);
15070 INPUT direcciones$(indicador,indice)
15080 NEXT indice
```

Ejecute el programa, después de entrar estas líneas, y seleccione la función 3. Todo lo que sucede a continuación, se debe exclusivamente a este bucle artificioso. Con sólo 4 instrucciones se consigue leer un completo juego de direcciones. Dentro del bucle se empieza emitiendo el nombre del campo. La variable 'indice' sirve de puntero al array de dichos nombres. Inmediatamente detrás del nombre de campo (¡punto y coma!) se lee el contenido del campo. Aquí se realizan dos indexaciones, puesto que las direcciones se almacenan en un array bidimensional. El primer índice resulta de la variable 'indicador', que es incrementada en 1 cada vez que se entra una dirección. Como segundo índice volvemos a utilizar la variable 'indice'. Por lo tanto, a través del teclado se van entrando 7 campos, uno tras otro.

Después de haber entrado los campos, debemos darle una oportunidad al usuario, de repetir la entrada. Es posible, que se haya equivocado en algún lugar y quiera corregirse.

```
15090 LOCATE 1,18
15100 PRINT "Datos entrados correctamente (s/n)";
15110 INPUT respuesta$
15120 IF respuesta$="s" THEN 15150
15130 IF respuesta$="n" THEN indicador=indicador-1:
GOSUB 300:GOTO 15000
15140 error$="pulse 's' o 'n'!":GOSUB 200:GOTO 15090
```

Si los datos han sido entrados correctamente, el programa prosigue su ejecución. Si los datos no han sido entrados correctamente, se vuelve a ejecutar el subprograma en cuestión. Pero previamente se reduce en 1 el contador de registros, para evitar que se active un nuevo registro en la línea 15030. Sin embargo, queremos volver a entrar el último registro. Si la respuesta dada es incorrecta, se emitirá un mensaje de error, y a continuación se realiza una lectura nueva.

El programa continuará ahora con otra pregunta.

```
15150 LOCATE 1,20
15160 PRINT "Mas entradas (s/n)";
15170 INPUT respuesta$
15180 IF respuesta$="s" THEN GOSUB 300:GOTO 15000
15190 IF respuesta$="n" THEN 1000
15200 error$="pulse 's' o 'n'!": GOSUB 200:GOTO 15150
```

En este momento, el usuario deberá decidir si desea efectuar otra entrada o no. En caso afirmativo, se vuelve a ejecutar el subprograma; en caso contrario, el programa vuelve al menú. Unà entrada errónea conducirá a otro mensaje de error y a una nueva entrada.

Con esto hemos finalizado el primer subprograma. Ahora ya podemos entrar cualquier cantidad de datos, aunque todavía no nos sirvan de nada. Para finalizar, le mostramos nuevamente el completo subprograma "ENTRAR DIRECCIONES", para que pueda compararlo con el suyo:

```
15000 REM ======================
15010 REM   ENTRAR DIRECCIONES
15020 REM ======================
15030 indicador=indicador+1
15040 PRINT
15050 FOR indice=1 TO 7
15060 PRINT campo$(indice);
15070 INPUT direcciones$(indicador,indice)
15080 NEXT indice
15090 LOCATE 1,18
15100 PRINT "Datos entrados correctamente (s/n)";
15110 INPUT respuesta$
15120 IF respuesta$="s" THEN 15150
15130 IF respuesta$="n" THEN indicador=indicador-1:
GOSUB 300:GOTO 15000
15140 error$="pulse 's' o 'n'!":GOSUB 200:GOTO 15090
15150 LOCATE 1,20
15160 PRINT "Mas entradas (s/n)";
```

```
15170 INPUT respuesta$
15180 IF respuesta$="s" THEN GOSUB 300:GOTO 15000
15190 IF respuesta$="n" THEN 1000
15200 error$="pulse 's' o 'n'!": GOSUB 200:GOTO 15150
```

MODIFICAR DIRECCIONES

Ahora queremos brindarle al usuario de nuestro programa la posibilidad de modificar campos aislados de un registro determinado. Para ello utilizaremos las teclas de función. Con la tecla **'v'** vamos a "hojear" hacia adelante y con **'r'** hacia atrás. Si se trata de modificar una dirección, debe pulsarse la tecla **'a'**. Con ayuda de la tecla 'RETURN' se vuelve al menú y, por lo tanto, se cierra la modificación.

Todo esto suena muy bien, pero falta programarlo. Empecemos por las primeras líneas:

```
20000 REM ======================
20010 REM  MODIFICAR DIRECCIONES
20020 REM ======================
```

Estas líneas no requieren ningún comentario. Pero ahora viene lo difícil. ¿Cómo solucionamos este problema? En primer lugar escogemos una variable, que debe representar el primer índice del array (tabla) de direcciones. Se trata de la variable **'contador'**, cuyo valor inicial 1.

```
20030 contador=1
```

Ahora ya podemos emitir la primera dirección de la tabla. Ya sabemos, que está compuesta por 7 campos, cuyos nombres están almacenados en un array. Volvemos a craer un bucle, que emita el número del campo, su nombre y el contenido.

```
20040 LOCATE 1,10
20050 FOR indice=1 TO 7
20060 PRINT indice;campo$(indice);direcciones$(contador,
indice)
20070 NEXT indice
```

Por consiguiente, se emitirán 7 líneas, conteniendo el número del campo, su designación y su contenido. El contenido del campo resulta del puntero **'contador'**, que señala al registro, y del propio indicador de campo (**'indice'**).

Una vez entrada la primera dirección, el usuario deberá tomar su decisión. Dispone de cuatro opciones:

tecla 'v'	**hojear hacia delante (dirección siguiente)**
tecla 'r'	**hojear hacia atrás (dirección anterior)**
tecla 'a'	**modificar dirección señalada**
tecla 'RETURN'	**volver al menú**

Ahora debemos leer el teclado hasta que se haya pulsado alguna tecla de función. El comando correspondiente es INKEY$.

Problema: **Leer un carácter del teclado**

Instrucción: **s=INKEY$**

Parámetros: s - **variable de cadena, que debe memorizar la tecla pulsada**

Ejemplo: **X$=INKEY$**
Comprueba, si se ha pulsado una tecla y memoriza dicha tecla en X$. Después continúa inmediatamente la ejecución del programa

Observación: **Si no se pulsa ninguna tecla, se borra la variable de cadena.**

Si desea, que el programa únicamente continúe después de haber pulsado una tecla, es necesesario entrar, por ejemplo,

la siguiente línea (por favor, no entre esta línea):

```
10 X$=INKEY$:IF X$="" THEN 10
```

En consecuencia, se va ejecutando la línea 10 una y otra vez, hasta que se pulse alguna tecla. Nuestro programa también constará de una instrucción de este tipo.

```
20080 tecla$=INKEY$:IF tecla$="" THEN 20080
```

Si el programa ha pasado por esta línea, será porque se ha pulsado alguna tecla.Ahora tan sólo debemos determinar, cuál es la tecla pulsada, y reaccionar en consecuencia. En primer lugar consultaremos la tecla RETURN, cuyo código ASCII es 13:

```
20090 IF tecla$=CHR$(13) THEN 1000
```

Por lo tanto, saltamos hacia el menú, si la tecla pulsada ha sido RETURN. En cambio, si se pulsa la tecla 'v', se emitirá el siguiente registro de datos. Para ello incrementamos el puntero **'contador'** en 1, pero únicamente en el caso en que **'contador'** no haya alcanzado todavía el último registro (contador=indicador). La línea siguiente se encarga de esta operación:

```
20100 IF tecla$="v" AND contador<indicador THEN
contador=contador+1:GOSUB 300:GOTO 20040
```

Unicamente en el caso de que se haya pulsado la tecla **'v'** Y todavía no se haya llegado a la última dirección, el contador será activado a la dirección siguiente, lo cual será emitido.

Al pulsar la tecla **'r'** ocurre algo parecido, con la diferencia de que **'contador'** se reducirá una unidad, siempre y cuando **'contador'** todavía no haya alcanzado el valor 1, porque eso significaría, que se trata de la primera direccion.

```
20110 IF tecla$="r" AND contador>1 THEN contador=contador-1:
GOSUB 300:GOTO 20040
```

En la línea 20040 se emite la dirección de puntero **'contador'**. Estos contadores han sido reducidos una unidad.

Ahora sólo queda comprobar, si se ha pulsado la tecla **'a'**. Con esta tecla se inicia la modificación.

```
20120 IF tecla$="a" THEN 20140
20130 error$="Error de entrada!":GOSUB 200:GOTO 20080
```

LLegamos a la línea 20130, si las teclas **'v'** o **'r'** iban a rebasar el límite del fichero, o si no se ha pulsado una tecla válida. En ese caso, simplemente se ignoran las teclas, se emite un mensaje de error y se efectúa un salto hacia la línea 20080, donde se realiza una nueva consulta del teclado.

Ahora ya sólo falta la propia modificación del registro a partir de la línea 20140. Allí se preguntará primero por el número, y después por el nuevo contenido del campo a modificar.

```
20140 LOCATE 1,18
20150 INPUT "Numero del campo (1-7): ";numero
20160 IF numero>0 AND numero<8 THEN 20180
20170 error$="Indicar numero 1-7!":GOSUB 200:GOTO 20140
20180 INPUT "Nuevo contenido: ";direcciones$(contador,numero)
20190 GOSUB 300:GOTO 20040
```

Esto ya ha sido el resto de nuestra rutina. En primer lugar, **'numero'** lee el número del campo a modificar. Después se comprueba, si **'numero'** contiene un valor válido. En caso afirmativo, la rutina procede a leer -a partir de la línea 20180- el nuevo contenido de este campo y emite el nuevo registro de direcciones a partir de la línea 20040. Si el número entrado no ha sido correcto, la línea 20170 emite un mensaje de error. Sólo se puede salir de esta rutina con ayuda de la tecla 'RETURN'. A continuación, le mostramos de nuevo el listado completo de la rutina "MODIFICAR DIRECCIONES":

```
20000 REM ========================
20010 REM  MODIFICAR DIRECCIONES
20020 REM ========================
20030 contador=1
20040 LOCATE 1,10
20050 FOR indice=1 TO 7
20060 PRINT indice;campo$(indice);direcciones$(contador,
indice)
20070 NEXT indice
20080 tecla$=INKEY$:IF tecla$="" THEN 20080
20090 IF tecla$=CHR$(13) THEN 1000
20100 IF tecla$="v" AND contador<indicador THEN
contador=contador+1:GOSUB 300:GOTO 20040
20110 IF tecla$="r" AND contador>1 THEN contador=contador-1:
GOSUB 300:GOTO 20040
20120 IF tecla$="a" THEN 20140
20130 error$="Error de entrada!":GOSUB 200:GOTO 20080
20140 LOCATE 1,18
20150 INPUT "Numero del campo (1-7): ";numero
20160 IF numero>0 AND numero<8 THEN 20180
20170 error$="Indicar numero 1-7!":GOSUB 200:GOTO 20140
20180 INPUT "Nuevo contenido: ";direcciones$(contador,numero)
20190 GOSUB 300:GOTO 20040
```

BORRAR DIRECCIONES

Ahora utilizaremos una técnica diferente de la anterior para borrar direcciones, ya que usted querrá conocer el mayor número posible de técnicas de programación. Debemos indicar los apellidos de la dirección a borrar. Si los desconocemos, podemos hallarlos con ayuda de la función "HOJEAR" de la rutina "MODIFICAR DIRECCIONES".

Pero antes de empezar con esta rutina del programa, hay que comprobar, si existen datos en la memoria del ordenador. Confeccionaremos una rutina a partir de la línea 500 para emitir el mensaje de error. Veamos primero la rutina completa:

```
500 REM =====================
510 REM      No hay datos!
520 REM =====================
530 error$="No hay datos en la memoria!"
540 GOSUB 200
550 RETURN
```

Esta rutina es muy corta, pero muy efectiva. Sería conveniente efectuarla también en la rutina "MODIFICAR DIRECCIONES". ¡Es evidente, que si no existen direcciones, no podemos modificarlas! Añadiremos la siguiente línea:

```
20030 IF indicador=0 THEN GOSUB 500:GOTO 1000
```

Se comprueba, si existen datos en la memoria del ordenador. La variable **'indicador'** siempre contiene el último registro, y, con ello, la cantidad de direcciones almacenadas en la memoria. Si el valor de esta variable es 0, significa, que no se han entrado ni cargado datos. En ese caso emitiremos el mensaje de error **"No hay datos en la memoria!"** con ayuda de la rutina que empieza por la línea 500, y después volvemos al menú.

Empecemos ahora con nuestra nueva rutina. Ya se ha demostrado, que las primeras líneas son las más sencillas.

```
25000 REM ===================
25010 REM  BORRAR DIRECCIONES
25020 REM ===================
25030 IF indicador=0 THEN GOSUB 500:GOTO 1000
25040 LOCATE 1,10
25050 INPUT "Apellido 1: ";apellido1$
25060 INPUT "Apellido 2: ";apellido2$
```

En primer lugar comprobamos, si ya existen datos. En caso afirmativo, la rutina lee los dos apellidos de la dirección a borrar. Se almacenan estas dos entradas en **'apellido1$'** y en **'apellido2$'**. Ahora debemos buscar dichos nombres en toda la tabla de direcciones. No podemos utilizar un bucle FOR...NEXT en este caso, ya que éste será abandonado con toda seguridad antes de tiempo, justamente en el momento, en que se haya localizado la dirección. No es aconsejable interrumpir los bucles FOR...NEXT prematuramente, por la siguiente razón: el ordenador memoriza la dirección de comienzo del bucle. Tan sólo después de procesar todo el bucle como es debido, se vuelve a borrar dicha dirección. Si se abandona el bucle varias veces antes de tiempo, dicha zona de memoria se va llenando. La consecuencia de ello puede ser, que el ordenador quede colgado.
Vamos, pues, a crear un bucle controlado automáticamente.

```
25070 bucle=1
25080 IF direcciones$(bucle,2)=apellido1$ AND
direcciones$(bucle,3)=apellido2$ THEN 25110
25090 IF bucle<indicadpr THEN bucle=bucle+1:
GOTO 25080
25100 error$="Direccion inexistente!":GOSUB 200:GOTO 1000
```

Primero asignamos un valor inicial de 1 a la variable **'bucle'**. Dentro del bucle se comprueba, si la dirección indicada con **'bucle'** coincide con la que estamos buscando. El segundo índice 2 y 3 corresponde al primero y segundo apellido, que serán comparados con **'apellido1$'** y **'apellido2$'**. Si los nombres coinciden, el programa continúa su ejecución en la línea 25110. En la línea 25090 se incrementa en 1 la variable de bucle, siempre y cuando no haya alcanzado todavía el final de la tabla de direcciones.

Si se procesa por completo todo el bucle, es decir, si no se interrumpe por haber encontrado el nombre en cuestión, significa, que el nombre no ha sido encontrado. En este punto emitimos un mensaje y volvemos al menú.

Ahora debemos continuar el programa en el punto, hacia el cual se bifurca el programa cuando encuentra el nombre buscado. Este punto corresponde a la línea 25110. Aquí debe mostrarse nuevamente la dirección completa a borrar, y a continuación el usuario decidirá, si realmente desea borrarla o no.

```
25110 GOSUB 300:LOCATE 1,10
25120 FOR indice=1 TO 7
25130 PRINT campo$(indice);direcciones$(bucle,indice)
25140 NEXT indice
25150 LOCATE 1,18
25160 INPUT "Borrar direccion (s/n) ";respuesta$
25170 IF respuesta$="s" THEN 25200
25180 IF respuesta$="n" THEN 1000
25190 error$="pulse 's' o 'n'!":GOSUB 200:GOTO 25150
```

Para empezar, esta rutina crea una nueva pantalla (línea 25110). Después emite el registro de direcciones con ayuda del bucle de las líneas 25120 a 25140. Entonces pregunta, si realmente debe borrarse dicha dirección. Si no debe borrarla, (por todas partes hay personas irresolutas), la rutina se bifurca hacia el menú. En el caso contrario, la dirección será borrada a partir de la línea 25200. Echemos primero una mirada a las últimas líneas de este subprograma.

```
25200 FOR i1=bucle TO indicador-1
25210 FOR i2=1 TO 7
25220 direcciones$(i1,i2)=direcciones$(i1+1,i2)
25230 NEXT i2
25240 NEXT i1
25250 indicador=indicador-1
25260 GOTO 1000
```

Para borrar una dirección, no basta con borrar la correspondiente posición de la tabla. Si lo hiciésemos, se conservaría el tamaño de la tabla a pesar del borrado. Pero esto no es lo que queríamos conseguir. Todas las direcciones, que se encuentran detrás de la dirección borrada, deben retroceder una posición. Si borramos p. ej. la dirección de índice 5, cuando existen 7 direcciones en la memoria, la sexta dirección retrocede hacia la posición de la quinta, la cual queda borrada de esta forma. La última, o sea séptima dirección retrocede hacia la posición 6. Si ahora reducimos el número de direcciones a 6, es decir, le restamos una, la dirección de índice 5 se habrá borrado efectivamente.

Al programar este principio, aparece por primera vez el concepto de bucle indentado. Dentro del bucle de variable **'i1'** se encuentra el bucle **'i2'**. El bucle **'i1'** recorre todos los registros a partir del registro a borrar. Uno por uno, se van desplazando todos los campos. Una vez finalizado este proceso, el programa está preparado para procesar el próximo registro. El bucle exterior sólo se ejecuta hasta **'indicador-1'**, puesto que el último registro está indicado con **'i1+1'** en la línea 25220. Si analiza las líneas correspondientes, descubrirá con facilidad la técnica utilizada para este desplazamiento.

Ahora ya hemos terminado también la confección de esta rutina de nuestro programa, cuyo volumen ya es bastante considerable. Como es habitual, le mostramos el listado completo de la rutina:

```
25000 REM ====================
25010 REM  BORRAR DIRECCIONES
25020 REM ====================
25030 IF indicador=0 THEN GOSUB 500:GOTO 1000
25040 LOCATE 1,10
25050 INPUT "Apellido 1: ";apellido1$
25060 INPUT "Apellido 2: ";apellido2$
25070 bucle=1
25080 IF direcciones$(bucle,2)=apellido1$ AND
direcciones$(bucle,3)=apellido2$ THEN 25110
25090 IF bucle<indicador THEN bucle=bucle+1:GOTO 25080
```

```
25100 error$="Direccion inexistente!":GOSUB 200:GOTO 1000
25110 GOSUB 300:LOCATE 1,10
25120 FOR indice=1 TO 7
25130 PRINT campo$(indice);direcciones$(bucle,indice)
25140 NEXT indice
25150 LOCATE 1,18
25160 INPUT "Borrar direccion (s/n) ";respuesta$
25170 IF respuesta$="s" THEN 25200
25180 IF respuesta$="n" THEN 1000
25190 error$="pulse 's' o 'n'!":GOSUB 200:GOTO 25150
25200 FOR i1=bucle TO indicador-1
25210 FOR i2=1 TO 7
25220 direcciones$(i1,i2)=direcciones$(i1+1,i2)
25230 NEXT i2
25240 NEXT i1
25250 indicador=indicador-1
25260 GOTO 1000
```

EMITIR DIRECCIONES

Esta parte del programa no es precisamente una de las más fáciles. Se trata de sacar, por primera vez, direcciones por la impresora. Pero también queremos posibilitar la emisión por pantalla. Después de elegir, si la emisión deberá hacerse por pantalla o por impresora, usted debe entrar los conceptos de búsqueda. Pero vamos a empezar escribiendo las primeras líneas.

```
30000 REM =====================
30010 REM  EMITIR DIRECCIONES
30020 REM =====================
30030 IF indicador=0 THEN GOSUB 500:GOTO 1000
30040 LOCATE 1,10
30050 INPUT "Impresora o pantalla (i/p) ";respuesta$
30060 IF respuesta$="i" THEN periferico=8:GOTO 30090
30070 IF respuesta$="p" THEN periferico=0:GOTO 30090
30080 error$="pulse 'i' o 'p'!":GOSUB 200:GOTO 30040
```

Nuevamente comprobamos, si hay datos en la memoria (línea 30030). Después la rutina pregunta, si debe emitir por impresora o por pantalla. La respuesta ('i' o 'p') es recogida en la variable de cadena **'respuesta$'**. En el caso de elegir la impresora, la variable **'periferico'** es activada a 8. Esto será necesario para el nuevo comando PRINT, describiremos a continuación.

Problema:	**Salida a un periférico**
Instrucción:	**PRINT #g,........**
Parámetros:	g - **Número del periférico (0-9)**
	0 - **pantalla**
	1 a 7 - **Ventana de pantalla (no explicada en este libro)**
	8 - **Impresora**
	9 - **Diskette**
Ejemplo:	**PRINT #8, "LISTA DE DIRECCIONES"** **Saca el texto "LISTA DE DIRECCIONES" por la impresora.**
Observación:	**El complemento '#g' también puede ser utilizado en relación con los ya conocidos comandos LIST e INPUT.**

Por consiguiente, con ayuda de PRINT también podemos sacar datos por la impresora o por la unidad de disco. En la variable **'periferico'** hemos almacenado el número correspondiente. Si ahora siempre indicamos **'PRINT #periferico'** para efectuar cualquier salida, los datos se emitirán por impresora o por pantalla, según el valor de la variable.

¿Pero por qué podemos indicar además el número de periférico junto con los comandos LIST e INPUT? En el caso del comando LIST, interesa la emisión del listado por impresora. Sin embargo, hay que tener en cuenta, que el complemento **'#8'** debe indicarse DETRAS del área numérico. Veamos dos ejemplos:

LIST #8 **saca el programa completo por la impresora**

LIST 1000-,#8 **saca todas las líneas por la impresora, a partir de la 1000 inclusive**

El complemento **'#9'** del comando INPUT lee datos del diskette, pero eso ya lo veremos más adelante. Ahora volvamos a nuestro programa.

Hasta ahora hemos definido el número del periférico para efectuar la salida. Sin embargo, habrá casos en los que no querremos emitir todas las direcciones, razón por la cual vamos a definir un criterio de búsqueda para cada campo. Observe en primer lugar las líneas siguientes:

```
30090 GOSUB 300:LOCATE 1,10
30100 PRINT"Conceptos de busqueda:"
30110 PRINT"----------------------"
30120 PRINT
30130 FOR indice=1 TO 7
30140 busca$(indice)=""
30150 PRINT campo$(indice);
30160 INPUT busca$(indice)
30170 NEXT indice
```

La línea 30090 crea ahora una nueva pantalla, antes de que las líneas 30100-30110 nos indiquen, que debemos entrar los conceptos de búsqueda. De nuevo, la entrada se efectúa con un bucle. Pero no sin borrar previamente los conceptos de búsqueda, que están almacenados en el array **'busca$'**, puesto que pueden contener todavía datos procedentes de búsquedas anteriores. A continuación, la línea 30160 lee los conceptos. Si se trata de buscar, por ejemplo, todas las personas cuyo primer apellido es García, sólo debe entrar RETURN en el primer campo. En el campo 2, que contiene el primer apellido, escribimos **"GARCIA"**, seguido de un RETURN. Los demás campos también deben ignorarse simplemente con RETURN. Pero si desea buscar además todos los "José García" del fichero, también debe escribir **"JOSE"** en 'Nombre'. Así pues, también podemos buscar determinadas direcciones por varios conceptos al mismo tiempo.

Tras la entrada del concepto de búsqueda, empezará la propia búsqueda.

```
30180 FOR i1=1 TO indicador
30190 hallado=0
30200 FOR i2=1 TO 7
30210 IF busca$(i2)="" OR busca$(i2)=direcciones$(i1,i2)
THEN hallado=hallado+1
30220 NEXT i2
```

Habrá observado otro bucle indentado, que regista todos los campos de los registros. Después de iniciarse el bucle exterior (línea 30180), en la línea 30190 se activa la variable **'hallado'** a 0. Esta variable nos servirá para contar aquellos campos, para los cuales o no habíamos entrado un concepto de búsqueda, o habíamos entrado un concepto que no coincide con el contenido del campo. En el caso de no entrar ningún concepto de búsqueda para algún campo, es decir, si sólo se ha pulsado RETURN, la variable **'busca(i2)'** no contiene nada (**busca(i2)=""**). La comparación y el incremento de la variable **'hallado'** se efectúa en la línea 30210. Si el valor del contador de éxitos **'hallado'** es 7 después de efectuarse el bucle interior (**i2**), la dirección responde a los criterios de la búsqueda y podrá ser emitida. Las líneas siguientes inician esta emisión:

```
30230 IF hallado<>7 THEN 30300
30240 IF periferico$=0 THEN GOSUB 300
30250 PRINT #periferico
```

Aquí se prepara la emisión del registro hallado. Unicamente si el registro no corresponde a los conceptos de búsqueda (**'hallado'** distinto de 7), se salta toda la emisión. Si se ha elegido la pantalla, se crea un nuevo encabezamiento. A continuación se saca una línea vacía en el periférico seleccionado. Ahora veamos las restantes líneas de este subprograma, que serán explicadas a continuación.

```
30260 FOR indice=1 TO 7
30270 PRINT #periferico, campo$(indice);direcciones$
(i1,indice)
30280 NEXT indice
30290 PRINT #periferico
30295 If periferico=0 THEN INPUT "Pulse RETURN ";respuesta$
30300 NEXT i1
30310 GOSUB 300:LOCATE 1,18
30320 PRINT "** FIN DEL FICHERO **"
30330 INPUT "Pulse RETURN ";respuesta$
30340 GOTO 1000
```

Estas líneas se ocupan casi exclusivamente de la emisión del registro hallado. Las líneas 30260 a 30280 forman un bucle, que envía los 7 campos de la dirección, junto con sus denominaciones, al periférico seleccionado.

Si se ha elegido la emisión por pantalla, no se prosigue la búsqueda hasta que se haya pulsado RETURN (línea 30295). En este caso, continuar la búsqueda significa incrementar el bucle 'i1' en 1 con el comando '**NEXT i1**'. Si se trata de emitir por impresora, la rutina se salta la línea que permite entrar RETURN. Por lo tanto, se imprimirán todas las direcciones una tras otra.

Finalizado el bucle 'i1', la búsqueda ha terminado, lo cual se indica visualizando el mensaje "**** FIN DEL FICHERO ****". Si el usuario pulsa ahora RETURN, se finaliza el subprograma, que se bifurca de nuevo hacia el menú.

Ya hemos superado otro obstáculo en el camino hacia nuestro programa de gestión de direcciones. Ahora sólo nos queda escribir las rutinas de carga y grabación del fichero en diskette.

El siguiente listado completo le ayudará a verificar su rutina "EMITIR DIRECCIONES":

```
30000 REM ====================
30010 REM  EMITIR DIRECCIONES
30020 REM ====================
30030 IF indicador=0 THEN GOSUB 500:GOTO 1000
30040 LOCATE 1,10
30050 INPUT "Impresora o pantalla (i/p) ";respuesta$
30060 IF respuesta$="i" THEN periferico=8:GOTO 30090
30070 IF respuesta$="p" THEN periferico=0:GOTO 30090
30080 error$="pulse 'i' o 'p'!":GOSUB 200:GOTO 30040
30090 GOSUB 300:LOCATE 1,10
30100 PRINT"Conceptos de busqueda:"
30110 PRINT"----------------------"
30120 PRINT
30130 FOR indice=1 TO 7
30140 busca$(indice)=""
30150 PRINT campo$(indice);
30160 INPUT busca$(indice)
30170 NEXT indice
30180 FOR i1=1 TO indicador
30190 hallado=0
30200 FOR i2=1 TO 7
30210 IF busca$(i2)="" OR busca$(i2)=direcciones$(i1,i2) THEN
hallado=hallado+1
30220 NEXT i2
30230 IF hallado<>7 THEN 30300
30240 IF periferico$=0 THEN GOSUB 300
30250 PRINT #periferico
30260 FOR indice=1 TO 7
30270 PRINT #periferico, campo$(indice);direcciones$
(i1,indice)
30280 NEXT indice
30290 PRINT #periferico
30295 If periferico=0 THEN INPUT "Pulse RETURN ";respuesta$
30300 NEXT i1
30310 GOSUB 300:LOCATE 1,18
30320 PRINT "** FIN DEL FICHERO **"
30330 INPUT "Pulse RETURN ";respuesta$
30340 GOTO 1000
```

GRABAR FICHEROS

Nos vamos acercando al final de la gestión de direcciones. Naturalmente, al apagar el ordenador, se pierden todos los datos que hemos entrado. Vamos a grabar nuestras direcciones en una unidad de diskette. ¿Qué precauciones deben tomarse al principio? Tal y como hemos venido haciendo hasta ahora, iniciamos esta rutina con su presentación en pantalla, verificando después la existencia de datos en la memoria.

```
10000 REM ==================
10020 REM   GRABAR FICHERO
10030 REM ==================
10040 IF indicador=0 THEN GOSUB 500:GOTO 1000
```

Estas líneas ya no son nada nuevo para nosostros. Antes de poder grabar los datos en diskette, hay que insertarla previamente. Las siguientes líneas nos invitan a hacerlo:

```
10050 LOCATE 1,12
10060 PRINT "Por favor, inserte diskette de datos! "
10070 PRINT
10080 INPUT "Despues pulse RETURN ";respuesta$
```

Estas líneas también son fácilmente comprensibles. Pero ahora empieza lo bueno. Si queremos escribir los datos con la instrucción **'PRINT #9,...'** en el diskette, debemos preparar previamente la unidad de disco. Se dice que "hay que abrir el fichero", utilizando la siguiente instrucción:

Problema: **Abrir un fichero para escritura**

Instrucción: **OPENOUT** "nombre"

Parámetros: nombre - **Nombre del fichero**

Ejemplo: **OPENOUT "direcciones"**
Abre el fichero "direcciones" para escritura

Así pues, abrimos nuestro fichero, antes de transmitir los datos.

Puesto que siempre utilizaremos el nombre **direcciones** para nuestro fichero, debemos borrar cualquier fichero del mismo nombre que esté grabado en el diskette. El proceso de borrar y abrir un fichero es realizado por las siguientes líneas del programa:

```
10090 a$="direcciones": |ERA,@a$
10100 OPENOUT "direcciones"
```

El carácter '|' se consigue pulsando SHIFT @.

Ahora ya podemos almacenar los datos. Lo primero que escribimos en el diskette es la cantidad de registros a transferir, que está memorizada en **'indicador'**. Después transferimos cada uno de los campos de la tabla de direcciones con ayuda de un bucle indentado, que ya conocemos sobradamente. Empezamos con los campos 1 a 7 del primer registro, seguimos con los campos 1 a 7 del segundo, y así sucesivamente:

```
10110 PRINT #9,indicador
10120 FOR i1=1 TO indicador
10130 FOR i2=1 TO 7
10140 PRINT #9,direcciones$(i1,i2)
10150 NEXT i2
10160 NEXT i1
```

Después de grabar los datos, hay que volver a cerrar el fichero. El comando adecuado para efectuar tal operación se denomina **'CLOSEOUT'**. Escribiremos la línea siguiente.

```
10170 CLOSEOUT
```

Una vez finalizado el almacenamiento de datos, queremos que se informe sobre tal hecho. Para ello escribimos las líneas siguientes:

```
10180 GOSUB 300
10190 PRINT "Las direcciones estan almacenadas!"
10200 FOR i=1 TO 2000:NEXT i1
10210 GOTO 1000
```

En primer lugar, se crea una nueva pantalla. Tras la indicación sigue un bucle de espera de unos 2 segundos, y después se produce un salto hacia el menú.

Así de sencillo es el almacenamiento de datos con la unidad de disco. Pero ahora veamos el modo de volver a recuperarlos.

CARGAR FICHEROS

El proceso de carga de las direcciones es casi idéntico al de su grabación. En este caso no se comprueba, si existen datos en la memoria. Por lo tanto es aconsejable, que el usuario grabe las nuevas direcciones que acabe de entrar, antes de cargar un nuevo fichero.

```
5000 REM ==================
5010 REM   CARGAR FICHERO
5020 REM ==================
5030 LOCATE 1,12
5040 PRINT "Por favor, inserte diskette!"5050 PRINT
5060 INPUT "Despues pulse RETURN ";respuesta$
5070 PRINT
```

Después de haber tomado todas las precauciones, podemos proceder a abrir el fichero, pero esta vez para la lectura. Vea, en primer lugar, la descripción del comando:

Problema: **Abrir un fichero para lectura**

Instrucción: **OPENIN** "nombre"

Parámetros: nombre - **Nombre del fichero**

Ejemplo: **OPENIN "direcciones"**
Abre el fichero "direcciones" para lectura

Ahora vamos a introducir este comando en nuestro programa:

```
5080 OPENIN "direcciones"
5090 INPUT #9,indicador
5100 FOR i1=1 TO indicador
5110 FOR i2=1 TO 7
5120 INPUT #9,direcciones$(i1,i2)
5130 NEXT i2
5140 NEXT i1
5150 CLOSEIN
```

Aquí se abre el fichero "direcciones" del diskette para la lectura. El acceso de lectura viene definido por el comando 'OPENIN'. Después se lee la cantidad de datos almacenados en este fichero. Esta variable será utilizada como valor final del bucle **'i1'**. Por lo tanto, se leerá la misma cantidad de registros que ha sido grabada previamente en diskette. La línea 5180 vuelve a cerrar el canal con el comando **'CLOSEIN'**.

```
50160 GOSUB 300:PRINT "Fichero esta cargado"
5170 FOR i=1 TO 2000:NEXT i
5180 GOTO 1000
```

Las últimas líneas de este apartado vuelven a emitir un mensaje y provocan un salto hacia el menú.

FINALIZAR PROGRAMA

Tal y como mencionamos al principio del presente capítulo, no es conveniente finalizar un programa, desconectando el ordenador de la red. A continuación mostraremos la forma de finalizarlo. Echemos nuevamente una vistazo a las primeras líneas.

```
35000 REM ======================
35010 REM   FINALIZAR PROGRAMA
35020 REM ======================
35030 IF indicador=0 THEN 35150
```

Seguramente ya habrá aborrecido las primeras líneas. Pero la línea 35030 parece nueva. Si 'indicador' es 0, es decir, cuando no hay datos en la memoria del ordenador, inmediatamente se finaliza el programa. Esto ocurre en la línea 35150.

```
5040 LOCATE 1,12
35050 PRINT "Todos los datos estan grabados (s/n) ";
35060 INPUT respuesta$
35070 IF respuesta$="n" THEN 1000
35080 IF respuesta$="s" THEN 35100
35090 error$="pulsar 's' o 'n'!":GOSUB 200:GOTO 35040
```

Poco a poco se va decubriendo el sentido de una rutina tal como "FINALIZAR PROGRAMA". Así pues, se trata de averiguar si todos los datos han sido grabados. Hay que tener en cuenta, que esta rutina puede haber sido llamada por equivocación. Si ahora contestamos con 'no', el programa vuelve al menú. Sin embargo, si las direcciones han sido grabadas correctamente, nada nos impide finalizar el programa.

```
35100 GOSUB 300:LOCATE 1,12
35110 PRINT "El programa puede ser ejecutado de"
35120 PRINT "nuevo con 'GOTO 1000', sin necesidad"
35130 PRINT "de perder datos!"
35140 PRINT
35150 END
```

Antes de finalizar el programa, queremos hacer una observación importante: una vez abandonado el programa, podemos volver a ejecutarlo con **'GOTO 1000'**, sin sufrir ninguna pérdida de datos. El comando RUN borra todas las variables antes de la ejecución, por lo cual su uso no es muy recomendable.

Ya lo hemos conseguido. Usted posee una gestión de direcciones documentada hasta el último detalle, en cuya confección habrá aprendido muchas cosas. Ahora podrá aplicar los conocimientos adquiridos para realizar sus propias ideas, o para cambiar programas -especialmente esta gestión de direcciones-, adaptándolos a su uso personal. Si no ha entendido inmediatamente alguno de los apartados, vuelva a analizarlo con tranquilidad. Si desea aprender a utilizar mejor el CPC-6128, le recomiendo la lectura de la extensa gama de libros DATA BECKER.

CAPITULO 6:

COLOR Y GRAFICOS

Un criterio de compra decisivo para gran parte de compradores de la amplia gama de ordenadores domésticos del mercado es precisamente su representación de colores y, sobre todo, su capacidad gráfica. El CPC, comparado con otros ordenadores de su clase, ofrece un colorido incomparable, así como excelentes propiedades gráficas. El propietario de un CPC6128 dispone de 27 colores, y puede elegir 2, 4 ó 16 colores, según la resolución grafica. La máxima resolución gráfica es de 640 x 200 puntos de pantalla, lo cual es realmente sensacional para un ordenador de su clase.

LOS MODOS OPERATIVOS GRAFICOS

El CPC6128 trabaja con tres resoluciones gráficas diferentes, seleccionables con el ya conocido comando 'MODE'. Por tanto, existe una estrecha relación entre la amplitud de caracteres y la resolución gráfica, lo cual queda reflejado en la tabla siguiente:

Comando	**caracteres/ línea**	**resolución gráfica**	**resolución caracteres**	**colores**
MODE 0	**20**	**160 x 200**	**32 x 8**	**16**
MODE 1	**40**	**320 x 200**	**16 x 8**	**4**
MODE 2	**80**	**640 x 200**	**8 x 8**	**2**

Este resumen permite apreciar claramente la relación entre amplitud de caracteres y resolución gráfica. También permite ver, que un carácter del **MODE 2** se compone de 8 por 8 puntos. En el **MODE 1**, esta matriz dobla su tamaño a 16 x 8 puntos. La mayor parte de los ordenadores de este tipo representan cada carácter con una matriz de 8 por 8 puntos al trabajar con 40 caracteres por línea. Por consiguiente podemos afirmar, que, en cualquier caso, el CPC6128 ofrece doble resolución que otros ordenadores de su clase.

En cada posición de la memoria se almacenan 8 puntos de la pantalla. La pantalla completa se compone de 640 x 200, o sea de 128.000 puntos, cuyo almacenamiento requiere 16.000 posiciones de memoria. Por lo tanto, la memoria gráfica ocupa unos 16 KByte de la memoria total (64 KByte). Al trabajar con las demás resoluciones, este espacio no se reduce, ya que, además del punto, debe almacenarse su color correspondiente. También se almacena texto normal en la memoria gráfica. Muchos ordenadores no juntan texto y gráficos, sino que disponen de memorias separadas para texto y gráficos.

LOS COLORES DE LA PANTALLA

Tras el encendido de su ordenador, observará los colores estándar del modo operativo **MODE 1**, azul y amarillo claro, suponiendo naturalemente, que usted haya conectado un monitor o televisor de color. Se trata de sólo dos de los 27 colores posibles. La siguiente tabla muestra todos los colores disponibles:

Número	Color	Número	Color
0	negro	13	blanco
1	azul	14	azul pastel
2	azul claro	15	naranja
3	rojo	16	rosa
4	magenta	17	magenta pastel
5	violeta claro	18	verde claro
6	rojo claro	19	verde marino
7	púrpura	20	cyan claro
8	magenta claro	21	verde limón
9	verde	22	verde pastel
10	cyan	23	cyan pastel
11	azul celeste	24	amarillo claro
12	amarillo	25	amarillo pastel
		26	blanco claro

Cuando se dispone de tal diversidad de colores, no es fácil encontrar un nombre para cada uno. Aparecen colores tales como blanco claro, denominación realmente absurda si no se

trata del anuncio de un detergente. Pero vamos a dejarnos de sutilezas.

Cada color se define por un número determinado, y no por su nombre. Por lo tanto, los números correspondientes a los colores estándar son el 1 de azul y 24 del amarillo.

LOS COLORES DEL MARGEN

Además de los colores de fondo y de caracteres, también podemos definir individualmente el color del margen. El margen es el área de la pantalla, que es inaccesible para el cursor. Tras el encendido del ordenador, no se puede distinguir el margen de la pantalla, ya que está representado con el mismo color que el fondo de la misma. Sin embargo, existe un comando que permite cambiar su color. Veamos en primer lugar la descripción del comando:

Problema: graduar el color del margen

Instrucción: BORDER col1,col2

Parámetros:
col1 - número del color
col2 - indicado tan sólo, si el color debe ir cambiando contínuamente entre 'col1' y 'col2'

Ejemplo:
BORDER 0
convierte el actual color del margen en negro
BORDER 0,13
el margen aparece cambiando entre los colores blanco y negro

Observación: ¡Es posible indicar uno cualquiera de los 27 colores en cualquier modo operativo (de MODE 0 a MODE 2)!

Ahora ya conoce el primer comando, que le permite inspeccionar los 27 colores. Por tanto, no esperemos más y vayamos a cambiar el color de los caracteres (número 24), utilizando la instrucción siguiente:

```
BORDER 24
```

Ahora puede distinguir el margen, que hasta este momento había permanecido oculto, y puede representarlo en cualquier color deseado. Pero somos muy vagos y no queremos repetir varias veces la misma instrucción. Por esta razón, vamos a confeccionar un programa, que se encargue de representar todos los colores, uno tras otro:

```
10 FOR color=0 TO 26
20 BORDER color
30 IF inkey$="" THEN 30
40 NEXT color
```

¿Cómo dice, no entiende este programa? En este caso se habrá saltado seguramente el capítulo de la introducción al BASIC, y habrá preferido este capítulo más interesante. Pero dudo que esto tenga algún sentido, porque irán apareciendo contínuamente pequeños programas, que servirán para ilustrarle la función de los nuevos comandos gráficos.

Pero volvamos ahora al programa. En un bucle creamos cada uno de los números de colores, que son asignados en la líneas 20. Al igual que todos los parámetros, el número del color también puede ser indicado como variable. La línea 30 espera a la pulsación de alguna tecla, para definir el siguiente color. De este modo puede dar un repaso rápido de todos los colores del CPC6128. Si está en posesión de un monitor verde, no podrá apreciar más que diferentes tonalidades del color verde. Sin embargo, esto no debería impedirle a investigar las posibilidades gráficas.

En la descripción del comando habrá observado, que el margen de la pantalla puede parpadear. Para ello deben indicarse dos números, separados por coma, detrás del comando. De esta forma, el margen va cambiando contínuamente entre estos dos colores, Probémoslo, utilizando los dos colores de mayor contraste, blanco y negro. La instrucción correspondiente es:

```
BORDER 0,26
```

Si quiere cuidar su vista, no es muy aconsejable continuar trabajando con un margen que parpadea de este modo. Vamos a desconectar el parpadeo.

BORDER 1

Una posible aplicación del parpadeo del margen puede ser el mensaje de error en un programa. De esta forma se comunica ópticamente al usuario la presencia de un error.

GRADUAR LA VELOCIDAD DE PARPADEO

Hay pocas cosas, que el CPC6128 sea incapaz de hacer. Así que también podemos graduar la velocidad de parpadeo del margen y de la pantalla. Para ello recurrimos al siguiente comando:

Problema: **Graduar la velocidad de parpadeo**

Instrucción: **SPEED INK time1,time2**

Parámetros:
time1 - **período del primer color**
time2 - **período del segundo color (siempre en unidades de 0.02 segundos)**

Ejemplo: **SPEED INK 50,100**
Mantiene el primer color durante un segundo, y el segundo color durante dos segundos

Observación: **'time1' y 'time2' pueden tomar valores comprendidos entre 1 y 255, lo que corresponde a un retardo de 0.02 a 5.1 segundos.**

Esta instrucción define pues el ritmo de parpadeo del margen. A continuación veremos otro programa ejemplo:

```
10 INPUT "1. color (0-26):";color1
20 INPUT "retardo (1-255):";ret1
30 INPUT "2. color (0-26):";color2
40 INPUT "retardo (1-255):";ret2
50 BORDER color1,color2
60 SPEED INK ret1,ret2
```

Aquí se definen ambos colores y su retardo, y las instrucciones siguientes provocan el parpadeo del margen.

CAMBIAR EL COLOR DE LOS CARACTERES

Ya sabemos, que el color estándar de los caracteres es el amarillo claro (número 24). Pero usted puede cambiar su color con ayuda de la siguiente instrucción:

Problema: **Cambiar el color de caracteres**

Instrucción: PEN #n,regcolor

Parámetros: #n - **Número de la ventana de pantalla (puede ser ignorado)**

regcolor - **Número del registro de color (0-15)**

Ejemplo: **PEN 1**

Cambia el color de los caracteres por el color contenido en el registro de color 1.

Observación: **Los registros de color deben modificarse con el cómando INK.**

Seguramente estará confundido con los registros de color. Usted ya sabe, que no puede utilizar todos los 27 colores al mismo tiempo. Según el modo operativo (comando MODE) podemos utilizar 2, 4 ó 16 de los 27 colores. En el modo de 40 caracteres podemos elegir entre 4 colores, que son dispuestos por el comando INK. Veamos la explicación de este comando:

Problema:	**determinar los colores seleccionados**	
Instrucción:	**INK** regcolor,color1,color2	
Parámetros:	regcolor -	**registro de color (0-15)**
	color1 -	**número del color (0-26)**
	color2 -	**sólo se indica, si debe ir cambiando continuamente entre los colores 'color1' y 'color2' (parpadeo).**
Ejemplo:	**INK 1,3** **Coloca el color rojo en el registro de color 1** **INK 3,1,9** **provoca, que el color del registro 2 vaya cambiando entre azul y verde.**	

Existen 16 registros de color numerados de 0 a 16, ya que podemos elegir un máximo de 16 colores (MODE 0). Así, por ejemplo, si queremos cambiar el color de los caracteres por el rojo, debemos colocar el número 3 (= rojo) en un registro de color. Si tomamos el registro 1, la instrucción correspondiente será:

```
INK 1,3
```

Ahora podemos cambiar el color de los caracteres con el comando 'PEN', indicando simplemente el anterior registro de color.

```
PEN 1
```

Esta instrucción ordena: "Da a los caracteres el color contenido en el registro 1". Previamente hemos depositado el color rojo en este registro. Al conectar el CPC, los registros ya contienen un color determinado. En la tabla siguiente se muestran los colores que corresponden a cada registro:

Registro de color	**MODE 0**	**MODE 1**	**MODE 2**
0	1	1	1
1	24	24	24
2	20	20	1
3	6	6	24
4	26	1	1
5	0	24	24
6	2	20	1
7	8	6	24
8	10	1	1
9	12	24	24
10	14	20	1
11	16	6	24
12	18	1	1
13	22	24	24
14	1,24	20	1
15	16,11	6	24

Puesto que en **MODE 1** y **MODE 2** la elección de colores es limitada, los colores se repiten en los registros. Así, en el MODE 1 por ejemplo, con el registro 1 accedemos simultáneamente a los registros de color 4, 8 y 12. Por esta razón es imposible elegir más de cuatro colores en el **MODE 1**, y más de dos colores en el **MODE 2**.

El comando INK provoca además, que todos los caracteres representados con el color contenido anteriormente en el registro, también cambien simultáneamente de color. Veamos un ejemplo: El color estándar de los caracteres proviene del registro 1 tras el encendido del ordenador. Si ahora cambia el color del registro 1 después del encendido, también cambia simultáneamente el texto, representado con ese color. Entre, por ejemplo, la instrucción siguiente:

```
INK 1,13
```

Todos los caracteres de la pantalla adquieren inmediatamente el color blanco. Sin embargo, si usted escribe en la pantalla caracteres del color contenido en el bote 2, el cambio del color del registro 1 no produce ningún efecto.

La asignación estándar del fondo de la pantalla es el registro 0. Por consiguiente, si elige el color del registro 0 para la representación de los caracteres, éstos coinciden con el color del fondo. La consecuencia de ello será, que no podamos distinguirlos del fondo. Pruébelo usted mismo:

```
PEN 0
```

El cursor desaparece. Para volver a la normalidad, deberá entrar la siguiente instrucción "a ciegas":

```
PEN 1
```

Así vemos, pues, que el registro 1 corresponde al color estándar de los caracteres. Puesto que siempre se reserva un color para el fondo de la pantalla, en el MODE 1 podemos utilizar tres colores diferentes para los caracteres. El siguiente programa emite tres líneas de colores diferentes:

```
10 MODE 1
20 INK 1,12:INK 2,9:INK 3,13
30 PEN 1:PRINT "amarillo"
40 PEN 2:PRINT "verde"
50 PEN 3:PRINT "blanco"
```

Para los tres colores de caracteres utilizamos los registros de 1 a 3. En la línea 20 asignamos los colores amarillo, verde y blanco a los registros 1 a 3. A continuación se emiten las tres líneas con diferentes colores.

También podemos asignar dos colores a un solo registro, lo cual provoca entonces un parpadeo de ambos colores. En primer lugar, ponga el ordenador en su estado de conexión (**SHIFT-CTRL-ESC**). Ahora sustituimos el color de los caracteres (registro 1) por blanco/negro.

INK 1,0,13

Inmediatamente parpadean todos los caracteres de la pantalla. Incluso parpadean los caracteres que usted escriba a continuación, ya que corresponden al registro de color 1. Para conseguir, que la letra posterior sea independiente de la letra que ya se encuentra en la pantalla, es necesario entrar la instrucción siguiente:

PEN 2

A partir de ese momento, el color de los caracteres proviene del registro 2. Podemos inhibir el parpadeo, colocando el color 24 en el registro 1.

INK 1,24

De nuevo sigue un programa de demostración:

```
10 MODE 1
20 INK 2,1,26
30 INK 3,26,1
40 PEN 2:PRINT "IR Y VOLVER"
50 PEN 3:PRINT "IR Y VOLVER"
60 PEN 1
```

CAMBIAR EL COLOR DEL FONDO

Ahora ya sabemos cambiar el color del margen y de los caracteres. existe otro comando, que cambia también el color del fondo:

Problema:	**Cambiar el color del fondo**	
Instrucción:	**PAPER** #n,regcolor	
Parámetros:	#n -	**número de la ventana de pantalla (puede ser ignorado)**
	regcolor -	**registro de color (0-15)**
Ejemplo:	**PAPER 2** **Sustituye el color del fondo por el color contenido en el registro 2.**	
Observación:	**Para cambiar todo el fondo de la pantalla, el comando PAPER debe ir seguido de CLS.**	

El comando **PAPER** se parece al comando **PEN** en el hecho, de que ambos requieren la indicación de un registro, que contenga el color deseado. El registro 0 es la asignación estándar para el fondo. No es aconsejable indicar el registro 1, ya que contiene el color de los caracteres. En tal caso, la escritura volvería a ser invisible.

Pero utilicemos ahora el comando **PAPER**. Queremos, que el fondo aparezca de color rojo claro, que se encuentra habitualmente en el registro 3. Por consiguiente, entre la instrucción:

```
PAPER 3
```

Sin embargo, ahora no aparece todo el fondo de color rojo claro, sino únicamente el fondo de los caracteres escritos a continuación. Si queremos modificar el fondo completo, deberá limpiar la pantalla (**CLS**) después del comando **PAPER**.

También se puede conseguir, que el fondo parpadee, indicando en la instrucción **PAPER** un registro que contenga dos colores. Vamos a provocar un parpadeo del fondo con los colores blanco y negro, después de volver a poner el ordenador en su estado de conexión (**SHIFT-CTRL-ESC**). Para ello es necesario entrar los siguientes comandos, siguiendo el orden señalado:

```
INK 3,0,13
PAPER 3
CLS
```

La frecuencia del parpadeo puede ser modificada con el ya mencionado comando 'SPEED INK'. Si desea cambiar el color p.ej. cada segundo, escriba la instrucción:

```
SPEED INK 50,50
```

Recuerde: los valores de esta instrucción representan unidades de 0.02 segundos (50 x 0.02 = 1 segundo).

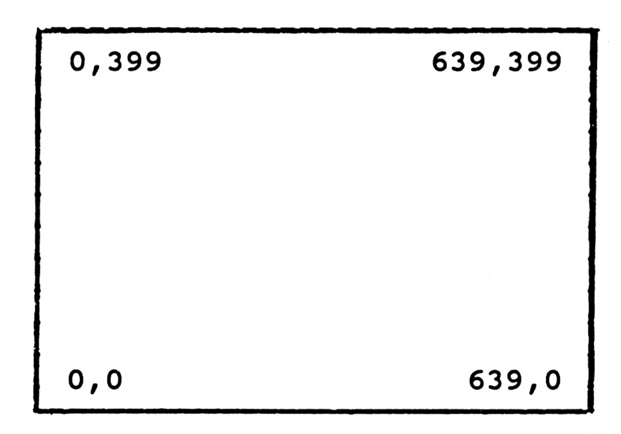
Para finalizar, veamos un pequeño programa ejemplo:

```
10 MODE 1
20 INK 2,0,13
30 INK 3,13,0
40 SPEED INK 50,50
50 PEN 2
60 PAPER 3
70 CLS
80 PRINT "POSITIVO Y NEGATIVO"
```

GRAFICOS DE ALTA RESOLUCION

Usted ya sabe, que puede elegir entre tres resoluciones gráficas diferentes, con distinto número de colores. Para dibujar figuras, existen algunos comandos, que únicamente serán presentados con el ejemplo de gráficos de alta resolución.

Recuerde: el modo gráfico de alta resolución está compuesto por 640 x 200 puntos de pantalla y cuatro colores, uno para el fondo y otro para los caracteres. Para poder acceder a cada punto, debemos imaginarnos la pantalla como un sistema de coordenadas. Al igual que en el comando LOCATE, que puede posicionar el cursor en cualquier punto de la pantalla, también debemos definir una posición en los comandos gráficos, utilizando dos valores. El primer valor define la posición del eje horizontal (eje X) de la pantalla, orientado de izquierda a derecha. El segundo valor corresponde a la posición del eje vertical (eje Y), contando de abajo a arriba. Puesto que la proporción de la resolución de 640 x 200 puntos no corresponde a la de la pantalla, el valor Y no se extiende de 0 - 199, sino de 0 - 399, y dos valores de Y definen una misma posición (0 y 1, 2 y 3, ...). Las cuatro esquinas tienen las coordenadas siguientes:



Después de familiarizarnos con este sistema, ya podemos pasar al estudio de los primeros comandos para la creación de gráficos.

TRAZADO DE PUNTOS

Problema: **Posicionar un punto**

Instrucción: **PLOT** x,y

Parámetros: x - **posición eje X (0-629)**
y - **posición eje Y (0-399)**

Ejemplo: **PLOT 0,0**
Posiciona un punto gráfico en la esquina inferior izquierda de la pantalla

En general, no es difícil entender los comandos gráficos si se domina el sistema de ejes de coordenadas. Pero el posicionamiento de puntos aislados empieza a ser aburrido, si no es combinado con otros comandos. Es muy interesante crear puntos dentro de un bucle. También es muy fácil trazar, por ejemplo, una línea. Aunque existe un comando especial para su trazado, este ejercicio constituye una práctica interesante del comando PLOT.

Ahora intente trazar de abajo a arriba una línea vertical en el extremo izquierdo de la pantalla. El siguiente programa realiza esta tarea:

```
10 MODE 2
20 FOR Y=0 TO 399
30 PLOT 0,Y
40 NEXT Y
```

Este programa es muy fácil de entender. Permítanos aumentar ligeramente el grado de dificultad. Escriba un programa, que

trace un marco alrededor de toda la pantalla. Vea la solución:

```
10 MODE 2
20 FOR Y=0 TO 399
30 PLOT 0,Y:PLOT 639,Y
40 NEXT Y
50 FOR X=0 TO 639
60 PLOT X,0:PLOT X,639
70 NEXT X
```

¡No habrá necesitado cuatro bucles! ¡Las dos líneas paralelas pueden ser trazadas simultáneamente con un único bucle! Pero esto son pequeños trucos, que se descubren pronto con la experiencia en la programación.

El trazado de líneas verticales puede ser acelerado considerablemente. ¡Recuerde, que cada dos valores consecutivos de Y definen un mismo punto! Eso significa, que podemos reducir el bucle Y a la mitad. Para ello aumentamos el incremento del bucle (Parámetro STEP) a 2, cambiando la línea siguiente:

```
20 FOR Y=0 TO 399 STEP 2
```

Observe la aceleración después de ejecutar nuevamente el programa. Ya ve, que es interesante analizar siempre, si no existe la posibilidad de acelerar el trazado del eje Y.

TRAZADO DE CIRCULOS

El CPC6128 no dispone de un comando específico para el trazado de un círculo, y tendremos que recurrir a otro procedimiento. Se trata de calcular y posicionar cada uno de los puntos del círculo, utilizando las formulas siguientes:

```
x = r * cos(a)
y = r * sin(a)
```

En lugar de 'r' debe indicarse el radio, y 'a' corresponde al ángulo. Podemos crear cada punto del círculo con ayuda de un bucle, que incremente el ángulo de 1 a 360. Observe, en primer lugar, el programa siguiente:

```
10 MODE 2
20 x=320:y=200:r=100
30 DEG
40 FOR a=1 TO 360
50 PLOT x+r*COS(a),y+r*SIN(a)
60 NEXT a
```

La línea 20 define el centro y el radio del círculo deseado. El comando DEG indica, que los posteriores cálculos de los ángulos se efectuarán en grados centesimales. El comando alternativo a éste es RAD, que calcula en radianes (grados sexagesimales). La equivalencia de **'90 grados sexagesimales = 100 grados centesimales'**. La línea siguiente representa el principio del bucle. Se trata de crear un círculo completo (360 grados). Si usted cambia los valores inicial y final, se generarán sólo sectores circulares. Ademas hay que sumar el centro (x e y) a la posición del punto hallado. A continuación llega el final del bucle.

Esta rutina le servirá ahora para trazar círculos, aunque no haya entendido la relación matemática. Si además desea trazar elipses, no puedo ayudarle aquí. Quizás sea conveniente, que saque del armario sus libros de la escuela, y que se dé clases particulares de geometría. Pero bromas aparte- la creación de gran parte de gráficos exige inevitablemente el uso de algoritmos matemáticos.

TRAZADO DE RECTAS

Problema: **Trazado de rectas**

Instrucción: **DRAW** x,y

Ejemplo: **DRAW 639,0**
Traza una recta entre la posición del cursor gráfico y la posición x/y indicada

La única duda, que puede surgir con este comando, es el cursor gráfico. Al principio, o sea después del encendido del ordenador, este cursor está posicionado en '0,0', y cualquier comando gráfico posterior lo actualiza. Por lo tanto, si posicionamos un punto en '100,200', el cursor gráfico también se hallará en esta posición. Un comando DRAW posterior traza la recta partiendo de este punto, y el último punto de la recta se convertirá en la nueva posición del cursor gráfico.

Sin embargo, existe la posibilidad de posicionar el cursor gráfico en cualquier otro lugar. El comando, que se explica a continuación, realiza esta tarea.

Problema: **Posicionar el cursor gráfico**

Instrucción: **MOVE** x,y

Ejemplo: **MOVE 200,250**
coloca el cursor gráfico en la posición '200,250'

Seguramente habrá observado, que, en las explicaciones de los dos últimos comandos, no hemos mencionado los parámetros. Esto se hizo intencionadamente, ya que actualmente todos conocemos el significado de 'x,y'.

Tracemos ahora una vertical en el margen izquierdo de la pantalla, tal como hicimos anteriormente con el comando PLOT. Para ello no necesitamos más que dos comandos, y después del encendido no más de uno:

```
10 MODE 2
20 MOVE 0,0
30 DRAW 0,399
```

Después del encendido del ordenador, no es necesario entrar la instrucción 'MOVE 0,0', porque en este caso el cursor gráfico ya tiene esta posición.

Ahora también vamos a trazar con DRAW el marco, que ha sido creado anteriormente con el comando PLOT:

```
10 MODE 2
20 MOVE 0,0: REM esquina inferior izquierda
30 DRAW 0,399: REM hacia esquina superior izquierda
40 DRAW 639,399: REM hacia esquina superior derecha
50 DRAW 0,639: REM hacia esquina inferior derecha
```

Este programa no precisa más explicación. La conexión entre puntos aislados ya no representa ningún problema. Pero lo que realmente despierta interés, es el trazado de líneas en relación con los bucles. Compruébelo usted mismo:

```
10 MODE 2
20 FOR i=0 TO 399 STEP 10
30 MOVE 399,i
40 DRAW 0,399-i
50 NEXT i
```

Se trata de un programa sencillo con un gran efecto. Y porque ha sido tan bonito, veamos otro ejemplo:

```
10 MODE 2
20 MOVE 0,0
30 FOR i=0 TO 399 STEP 10
40 DRAW 399,i
50 DRAW 399-i,399
60 DRAW 0,399-i
70 DRAW i,0
80 NEXT i
```

¿No es admirable cuando se generan tales imágenes en la pantalla, que además son de su propia creación? Quizás tenga la tentación de realizar su propias fantasías gráficas. Es muy divertido, aunque no salga de inmediato. En confianza, yo también me he calentado el coco creando esta imagen. La dificultad estriba en transportar la figura de nuestra imaginación al sistema de coordenadas, es decir, en crear el algoritmo. Esto acaba en verdaderos ejercicios mentales.

TRAZADO RELATIVO

Para finalizar este capítulo le mostramos una variante de los comandos explicados hasta ahora. Con los comandos PLOT, MOVE, el cursor gráfico no ha tenido ninguna importancia, porque siempre hemos definido un punto absoluto. Sin embargo, su significado cambia si añadimos la letra 'R' a dichos comandos:

```
PLOTR x,y
DRAWR x,y
MOVER x,y
```

En estos casos, no es necesario indicar las coordenadas x/y absolutas, sino un valor relativo. Para averiguar las coordenadas definitivas, simplemente se suma el valor X relativo, indicado en el comando, a la posición X del cursor gráfico. Lo mismo vale para la coordenada Y. Veamos un ejemplo: partimos de una posición inial del cursor gráfico en '200,100'. La instrucción 'PLOTR 50,50' coloca un punto en la posición '250,150'.

El siguiente ejemplo servirá para aclararlo un poco más. Queremos trazar un cuadrado de 10 por 10 puntos. Las coordenadas de la esquina superior izquierda se encuentran en las variables 'x' e 'y'.

```
10 MODE 2
20 x=200:y=100
30 MOVE x,y
40 DRAWR 10,0: REM 10 hacia la derecha
50 DRAWR 0,10: REM 10 hacia arriba
60 DRAWR -10,0: REM 10 hacia la izquierda
70 DRAWR 0,-10: REM 10 hacia abajo
```

Ha podido observar, que también hemos indicado número negativos. Esto es necesario para dirigirnos relativamente a las posiciones, que se encuentran a la izquierda o debajo de la posición gráfica.

Con esto finalizamos el capítulo de gráficos para principiantes. En el manual, o en otros libros del CPC de DATA BECKER, encontrará otros comandos gráficos.

CAPITULO 7:

GENERACION DE SONIDO

Un ordenador tiene muy buena capacidad para generar sonido. El procesador -el que realmente trabaja en un ordenador- es controlado por un generador de impulsos. Este generador genera aproximadamente 1-4 MHz (milliones de impulsos por segundo) según el ordenador. La solución más sencilla consiste en trasladar este impulso a un altavoz. Sin embargo, los sonidos generados de esta forma no son perceptibles para el oído humano, por lo cual debe reducirse considerablemente la frecuencia del sistema. Ello se consigue, partiendo la frecuencia antes de enviarla al altavoz. Las frecuencias generadas en este proceso son percibidas como sonidos.

Eso fue la primera generación de la creación de sonido con el ordenador. Pero estos sonidos, comparables con los sonidos de aviso de un reloj de pulsera, no resultaron muy satisfactorios, y se prosiguió desarrollando la generación de sonido en ordenadores domésticos. Como resultado de esta evolución, muchos ordenadores domésticos están provistos de un sintetizador, albergado en un Chip. Los sonidos generados de este modo se parecen sorprendentemente a los isntrumentos musicales "naturales".

El Chip que genera sonido en el CPC6128, no puede ser denominado sintetizador en el propio sentido de la palabra. Carece de propiedades importantes, tales como diversas formas de onda (senoidal, rectangular, etc.) y los filtros que influyen en el timbre. Tampoco se da el verdadero principio del sintetizador, es decir el hecho de que la frecuencia (altura) de los tonos dependa de una tensión. El generador de sonido del CPC6128 hace que la frecuencia dependa de un factor de división.

El generador de sonido del CPC6128 dispone de tres voces. Eso significa, que pueden sonar tres tonos al mismo tiempo. También dispone de un generador de ruido blanco, que puede generar ruidos además de tonos.

Cualquier sonido que proviene del generador, se emite a través del altavoz incorporado en el CPC. Por esta razón es necesario bajar el volumen del posiblemente conectado televisor. A la derecha del CPC se puede graduar el volumen del altavoz incorporado.

Pero la persona que desee escuchar sus propias creaciones en estéreo, tiene la posibilidad de conectar la salida de audio del CPC con un equipo estereofónico. Para ello, el CPC dispone de una hembrilla de salida (I/O) en la parte trasera izquierda. En este caso hay que fabricarse un cable, si no puede conseguirse ninguno en tiendas especializadas. La salida del CPC debe unirse con la entrada de su amplificador, que habitualmente está conectado a un cassette o un sintetizador. También puede utilizarse la hembrilla AUX.

En el siguiente capítulo le enseñaremos los comandos BASIC más importantes para la generación de sonido. Pero es imposible explicar aquí la completa generación de sonido. En el manual y en otros librs DATA BECKER hallará más información sobre este tema.

EL COMANDO SOUND

Problema: Determinar las características del sonido

Instrucción: SOUND a, b, c, d, e, f, g

Parámetros:

a - estado del canal (aquí siempre 1)
b - divisor, determina la frecuencia (0-4095)
frecuencia = 125000/divisor
c - Duración del sonido (-32768 a +32767, en unidades de 0.01 segundos)
d - volumen (0-15)
e - curva envolvente para el volumen (0-15)
f - curva envolvente para el sonido (0-15)
g - carácter de ruido blanco (0-15)

Ejemplo: **SOUND 1,1000,100,7,0,0,0**
genera un sonido de 125 Hz durante 1 segundo de volumen 7

Este comando es muy extenso, y no utilizaremos todos los parámetros. Especialmente la programación de las curvas envolventes rebasaría el volumen de este libro. Sin embargo, los conocimientos adquiridos en este capítulo ofrecen una buena base para seguir profundizando en el tema con ayuda del manual.

LA ALTURA DEL TONO

En primer lugar, vamos a explicar con más detalle los parámetros de mayor interés. El primer parámetros **'a'** indica un estado de canal, que, entre otras cosas, permite la emisión simultánea de tres voces. En los ejemplos siguientes trabajaremos sólo con una voz, la voz 1.

El siguiente parámetro **'b'** define la altura del tono. Aquí se indica un divisor, por el cual se divide la frecuencia básica de 125000 Hz. Se admiten frecuencias comprendidas entre 0 y 125000 Hz, ya que el divisor puede tomar un valor de 0 a 4095. El siguiente programa cambia gradualmente la frecuencia de un tono desde 30000 hasta 300 Hz. De esta forma podrá apreciar más fácilmente las diversas frecuencias.

```
10 FOR divisor=4.1 TO 416 STEP 0.1
20 PRINT INT(125000/divisor)
30 SOUND 1,divisor,1,7,0,0,0
40 NEXT divisor
```

La función **'INT'** separa los decimales del valor cerrado entre paréntesis, puesto que no son necesarios para realizar nuestro ejemplo. Ya habrá observado, que, al igual que ocurre en todos los comandos, en el comando SOUND también podemos sustituir los parámetros por variables.

La variación de la altura del tono en un bucle permite conseguir numerosos efectos. Así podemos simular, por ejemplo, una sirena:

```
10 FOR divisor=120 TO 200
20 SOUND 1,divisor,2,7,0,0,0
30 NEXT divisor
40 FOR divisor=200 TO 120 STEP -1
50 SOUND 1,divisor,2,7,0,0,0
60 NEXT divisor
70 GOTO 10
```

Esta sirena, que nos recuerda los coches de policía americanos, puede ser interrumpida con la tecla 'ESC' (pulsar dos veces). Para todos aquellos, que deseen continuar jugando con la sirena, se ofrece la siguiente variante del programa:

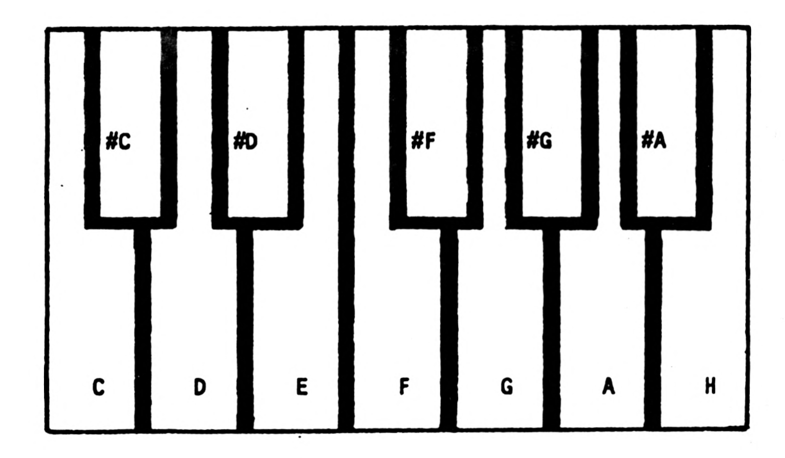
```
5 INPUT "divisor inicial ";a
6 INPUT "divisor final   ";b
7 INPUT "velocidad (1-10)";t
10 FOR divisor=a TO b
20 SOUND 1,divisor,t,7,0,0,0
30 NEXT divisor
40 FOR divisor=b TO a STEP -1
50 SOUND 1,divisor,t,7,0,0,0
60 NEXT divisor
70 GOTO 10
```

Aquí se asignan las características más destacadas de la sirena deseada a unas variables, que son utilizadas después en el programa. Al igual que antes, también puede interrumpir el programa con 'ESC'.

MUSICA A PARTIR DE FRECUENCIAS

Los instrumentos musicales no generan los tonos sin graduación. Sólo una graduación determinada hace el sonido agradable para el oído humano. Para entenderlo mejor, debemos hacer algunas aclaraciones básicas.

Una escala musical se compone de 12 tonos diferentes, tal como puede observar en la siguiente figura, que representa un teclado de piano.



Un teclado de piano se compone de varias secciones de este tipo. Estas secuencias de tonos se denominan octavas. La primera octava comprende los tonos más graves, y la última los tonos más agudos.

¿En cuántos grados se divide cada tono? El cálculo de las frecuencias de cada tono parte internacionalmente del tono **'A'** ("la") normal de la octava central, de 440 Hz. Veamos cómo suena este tono. ¿Qué divisor debemos utilizar para esta frecuencia? Recuerde: el divisor se obtiene a partir de la fórmula **'DIVISOR=125000/FRECUENCIA'**. Eso da un resultado de 284.090909 con una frecuencia de 440 Hz. Sin embargo, el divisor debe ser siempre entero, es decir, no debe tener decimales. De esta forma se produce una ligera variación de la frecuencia, que sólo podrá ser apreciada por un especialista en la materia. Genere ahora el tono 'A' central.

```
SOUND 1,284,100,7,0,0,0
```

Ahora falta saber, cómo podemos determinar los demás tonos de la frecuencia básica. La fórmula correspondiente está compuesta de la forma:

```
FRECUENCIA = 440 * (2 ^ (OCTAVA + (N-10) / 12))

OCTAVA -  de -3 a 4
N -       posición del tono en la escala musical
          ('C'=1, 'A'=12)
```

Dejaremos de lado el origen matemático de la fórmula. Lo importante es, que usted sepa utilizarla. Si se trata, por ejemplo, de generar el tono 'C' de la octava inferior, la frecuencia debe hallarse de la siguiente forma:

```
440 * (2^(-3+(1-10)/12)) = 32.7031957 Hz
```

Además habrá que determinar el divisor para poder generar el tono en cuestión. Ya conocemos la fórmula correspondiente:

```
12500/32.7031957 = 3822.25643
```

Por consiguiente, el divisor 3822 genera el tono 'C' de la octava inferior. De este modo podemos volver a crear un programa que toque la escala musical:

```
10 INPUT "octava (-3 a 4)";octava
20 FOR tono=1 TO 12
30 frecuencia=440*(2^(octava+(tono-10)/12))
40 divisor=INT(12500/frecuencia)
50 SOUND 1,divisor,20,7,0,0,0
60 NEXT tono
```

Cada vez resulta más interesante experimentar con el generador de sonido. Ahora vamos a colmar este capítulo: el siguiente programa "PIANO PARA PRINCIPIANTES" convierte su CPC en un instrumento musical.

```
10 MODE 0
20 DIM tono(255)
30 tono$="Q2W3ER5T6Y7UI90OP"
40 FOR i=1 TO 17
50     indice=ASC(MID$(tono$,i,1))
60     tono(indice)=i
70 NEXT i
80 CLS
90 PRINT "     PIANO PARA"
100 PRINT "     PRINCIPIANTES"
110 LOCATE 1,5
120 PRINT " 2 3   5 6 7   9 0"
130 PRINT
140 PRINT "Q W E R T Y U I O P"
150 a$=INKEY$:IF a$="" THEN 150
160     IF tono(asc(a$))=0 THEN 150
170     tono=tono(asc(a$))
180     frecuencia=440*(2^(tono/12))
190     divisor=INT(125000/frecuencia)
200     SOUND 1,divisor,30,7,0,0,0
210 GOTO 150
```

!Un pequeño programa de gran efecto! Probablemente usted efectúe algunas mejoras, cuando haya entendido el modo de funcionamiento.

RUIDO BLANCO

Al explicar el comando SOUND, ya hicimos alusión a un parámetro, que permite generar ruido blanco. Es el último de los parámetros, y debe contener un valor comprendido entre 1 y 15 si queremos activar el ruido. Por consiguiente, el comando SOUND permite generar tanto sonidos como ruido, según el valor de los parámetros 'b' y 'g'. Vea el siguiente cuadro:

SOUND a,b,c,d,e,f,g,			
Parámetros		Resultado	
'b'	'g'	sonido	ruido
0	0	no	no
1-4095	0	sí	no
0	1-15	no	sí
1-4095	1-15	sí	sí

Como puede ver, el comando SOUND permite generar sonido y ruido simultáneamente. El ruido puede graduarse modificando el valor del parámetro de 1 (ruido agudo) a 15 (ruido grave). El siguiente programa lo demuestra:

```
10 FOR r=1 TOO 15
20     SOUND 1,0,20,7,0,0,1
30 NEXT r
```

Podemos apreciar claramente los diversos ruidos. Hemos eliminado el sonido, desactivando el segundo parámetro.

El ruido blanco nos servirá ahora para provocar diversos efectos. Con ayuda de un generador de ruido blanco podemos crear ruidos tales como viento, disparos y explosiones. El siguiente programa imita a una locomotora:

```
10 SOUND 1,0,10,7,0,0,15
20 SOUND 1,0,5,7,0,0,0
30 GOTO 10
```

Puede detener la locomotora con ayuda de la tecla 'ESC'. El ruido (0.1 segundos) va seguido de una pausa (0.05 segundos). Recuerde: la duración del sonido se define a través del tercer parámetro, expresada en unidades de 0.01 segundos.

El último ejemplo debe provocar un disparo. Además se reduce el volumen con ayuda de un bucle.

```
10 a$=INKEY$:IF a$="" THEN 10
20 FOR v=7 TO 1 STEP -1
30     SOUND 1,0,10,v,0,0,15
40 NEXT v
50 GOTO 10
```

Al pulsar cualquier tecla, se producen los disparos. El programa puede ser interrumpido con la tecla 'ESC'. ¿Por qué no intenta generar diversos ruidos? Ahora ya conoce todos los comandos BASIC necesarios para hacerlo.

CAPITULO 8:

LA UNIDA DE DISCO

Seguramente ya se habrá percatado, de que su CPC6128 lleva una unidad de disco incorporada. Este dispositivo sirve para grabar sus datos y programas. Usted ya ha aprendido el simple grabado y carga de sus programas. En el programa de direcciones también ha podido practicar el almacenamiento de datos con la unidad de disco. Todo esto ya no lo veremos en el presente capítulo, que será dedicado a explicar facultades especiales, demostradas con ayuda de ejemplos.

LOS DISKETTES

Las unidades de disco de Amstrad se diferencian de las unidades habituales en un punto esencial, en el tamaño de los diskettes. Después de utilizar durante varios años diskettes de 5 1/4 pulgadas, parece ser que los diskettes de 3 1/2 pulgadas se conviertan en un nuevo estándar en el mercado de la informática. Los ordenadores de la nueva generación (AMIGA, 520ST, MacIntosh) también están equipados con este formato de diskettes. Las unidades de Amstrad, en cambio, trabajan con un formato de diskettes, que no puede hallarse en ningún otro equipo: el formato de 3 pulgadas.

Los diskettes del CPC tienen una ventaja comparados con los diskettes de 3 1/2 pulgadas; pueden ser grabados en ambas caras, es decir, pueden ser girados. De esta forma disponemos de una capacidad de almacenamiento de 2 por 180 KBytes en un diskette. Físicamente, el diskette está dividido en 40 pistas. Por su parte, cada una de ellas contiene 9 sectores. En cada uno de los 360 sectores pueden almacenarse 512 Bytes. De todo ello resulta una capacidad total de 180 KBytes. Pero no tema, esta estructura física no tiene ningún significado para usted. El sistema operativo, mejor dicho el DOS (Disk-Operating-System) le evita el trabajo de organizar la composición del diskette.

EL FORMATEADO

Antes de poder utilizar los diskettes como memoria de datos, es imprescindible formatearlos. Durante el formateado, se disponen los sectores en el diskette para recibir los datos. Veamos primero el procedimiento exacto para el formateado:

1. **Inserte la cara A del diskette de sistema adjunto (CP/M) en la unidad.**
2. **Llame al sistema operativo CP/M con el comando "|CPM" (El carácter "|" se obtiene pulsando SHIFT y @).**
3. **Cargue del diskette el programa para el formateado, escribiendo "DISCKIT3".**
4. **Pulse la tecla de función f7 para elegir el formateado.**
5. **Pulse la tecla de función f6 para elegir el formato de los datos.**
6. **Inserte el diskette a formatear.**
7. **Pulse la tecla Y.**
8. **Pulse cualquier tecla después de haberse finalizado el formateado.**
9. **Vuelva a insertar el diskette de sistema.**
10. **Abandone el programa auxiliar con la telc ade función f0**
11. **Ponga el ordenador en su estado de conexión (SHIFT-CONTROL-ESC)**

Este proceso es muy complicado, pero es inevitable. Si se ha efectuado correctamente, ya dispone de un diskette con una cara formateada, lista para ser utilizada como memoria de datos.

NOMBRES DE FICHERO

Al designar unos nombres para programas y ficheros en un diskette, debe tener las siguientes precauciones:

Los nombres de programa no deben exceder los 8 caracteres.

Dispone de tres caracteres más, si separa el nombre de fichero de estos tres caracteres mediante un punto.

Los programas grabados de forma habitual, reciben automáticamente el suplemento **.BAS,** si usted no indica ningún otro nombre adicional. Vea un ejemplo:

SAVE "TEST"	El programa es registrado en el diskette con el nombre de **TEST.BAS.**
SAVE "TEST.DEM"	Elprograma es registrado en el diskette con el nombre de **TEST.DEM.**

EL INDICE

Entre algunas líneas BASIC y grábelo como programa con el nombre de **TESTCAT.**

SAVE "TESTCAT"

En caso necesario, el índice del diskette puede visualizarse en la pantalla con ayuda del comando **CAT.**

CAT

Si usted entra ahora este comando, se mostrará el índice de la siguiente forma:

```
Drive A: user 0

TESTCAT .BAS 1K

177K free
```

De esta forma puede observar, que su programa no ocupa más de 1KByte, y que todavía dispone de 177 KBytes de memoria en el diskette.

Para cargar el programa, escriba la siguiente instrucción:

```
LOAD "TESTCAT"
      o bien
LOAD "TESTCAT.BAS"
```

A continuación explicaremos otras posibilidades respecto a la gestión de programas.

PROGRAMAS COMO FICHEROS ASCII

En el anterior transcurso del libro aparecieron las expresiones "fichero" y "programa", que siempre deben ser diferenciados claramente. ¿Pero cuál es la diferencia entre ambos? Los programas están dispuestos, tanto en el diskette como en otros medios de almacenamiento, en forma comprimida. Los comandos, por ejemplo, son almacenados de forma codificada, ya que, de lo contrario, el comando "PRINT" p.ej. ocuparía cinco posiciones de memoria. Para evitarlo, cada comando posee un código (0-255), que ocupa tan sólo una posición de la misma. Esto puede compararse p.ej. con los caracteres del CPC, que también son codificados en el interior del ordenador ('ASC' y 'CHR$').

El CPC ofrece la posibilidad de almacenar los programas escritos de forma completa, o sea en texto descifrado. Vea a continuación la descripción del comando correspondiente:

Problema: **Almacenar programas en texto descifrado**

Instrucción: **SAVE** "nombre"**,A**

Parámetros: nombre - **nombre del programa**

Ejemplo: **SAVE "direcciones",A**
Almacena el programa denominado "direcciones" en texto descifrado

Observación: **El programa almacenado puede ser leído como fichero, pero no como un programa ejecutable.**

Seguramente no tendrá idea del uso que puede tener este método de almacenamiento. No sabrá apreciar sus ventajas hasta que no trabaje con un procesador de texto. Los programas almacenados como fichero ASCII pueden ser leídos y procesados por un procesador de textos. Esto es especilamente interesante, cuando se trata de incorporar listados de programas en un texto, tal como también se practicó en este libro.
Con ayuda del siguiente ejemplo, usted puede convencerse de que estos programas pueden ser leídos como fichero. Para ello entre el siguiente programa de demostración:

```
10 REM ======================
20 REM  PROGRAMA COMO FICHERO
30 REM ======================
40 PRINT"El programa ha finalizado"
```

Almacene este programa con la siguiente instrucción en un diskette:

```
SAVE "DEMO.ASC",A
```

Ahora vamos a leerlo como si fuese un fichero, y lo visualizaremos en la pantalla. Para ello necesitamos este programa:

```
10 OPENIN "DEMO.ASC"
20 FOR i=1 TO 4
25 INPUT #9,a$
30 PRINT a$
40 NEXT i
50 CLOSEOUT
```

Al entrar y ejecutar este programa, observará sorprendido, que ahora se visualiza en la pantalla el programa de demostración almacenado anteriormente.

PROGRAMAS PROTEGIDOS

Quizás usted haya intentado alguna vez cargar en el ordenador un programa comprado, y listarlo después. Si no lo ha conseguido, es debido a que estos programas han sido protegidos. Usted también puede proteger sus programas si lo desea. La descripción del siguiente comando le explica el modo de hacerlo:

Problema: **Protección de programas**

Instrucción: **SAVE** "nombre",**P**

Parámetros: nombre - **nombre del programa**

Ejemplo: **SAVE "secret",P**
Graba el programa "secret" en el diskette de tal forma, que ya no podrá ser listado.

Observación: **Es aconsejable grabar una versión del programa protegido de forma normal, ya que la protección es irreversible.**

Esta descripción ya explica prácticamente todo lo que necesita para grabar programas protegidos. Vamos a probarlo inmediatamente, entrando las siguientes líneas BASIC:

```
10 REM ============
20 REM    secret
30 REM ============
40 PRINT " Entre el código:";
50 INPUT a$
60 IF a$<>"xy12" THEN NEW
70 PRINT "o.k."
```

Grabe este programa con la instrucción

```
SAVE "secret.p",P
```

Y ahora cargue el programa de nuevo con la instrucción

```
LOAD "secret.p"
```

Ejecute el programa con '**RUN**'. ¿Cómo, el programa no se ejecuta? No tema, aquí no ha fallado nada, porque los programas protegidos no pueden ser ejecutados con '**LOAD**' y '**RUN**', sin únicamente con '**RUN"**nombre**"**'. Por lo tanto, escriba ahora la instrucción correcta:

```
RUN "secret.p"
```

Esta es una posibilidad de cargar un programa y ejecutarlo al mismo tiempo.

Después de haber cargado el programa, será ejecutado automáticamente. ¿Y no es posible descifrar la clave? ¿Por qué no intenta interrumpir el programa con la tecla '**ESC**'? Pulse dicha tecla dos veces. El programa se ha detenido realmente. Pero no triunfe todavía, porque si ahora entra el comando '**LIST**', no sucede nada, ya que no queda ningún programa en la memoria del ordenador. Ha sido borrado al interrumpirlo con '**ESC**'. Por consiguiente, no hay posibilidad de abrir el programa. Pero esto no significa, que los conocedores experimentados del CPC tampoco tengan éxito al intentarlo. Con los adecuados conocimientos del lenguaje máquina, es posible abrir cualquier código por muy bueno que sea. Hagamos un breve resumen final:

El suplemento ',P' detrás del comando SAVE protege cualquier programa contra el listado no autorizado.

El programa sólo puede ser cargado y ejecutado con 'RUN"nombre"'.

Al detener el programa con 'ESC', se borrar la memoria del ordenador.

La protección es un proceso irreversible.

ALMACENAMIENTO DE SECTORES DE LA MEMORIA

La memoria completa del CPC comprende 2 por 64 KBytes, es decir 2 por 65535 Bytes (caracteres). Si queremos grabar sectores aislados de esta memoria, debemos utilizar la instrucción siguiente:

Problema: Almacenar sectores de la memoria.

Instrucción: SAVE "nombre",B,de,hasta,inicio

Parámetros:

nombre -	nombre del programa
de -	dirección inicial (0-65535)
hasta -	dirección final
inicio -	dirección de inicio (con programas en lenguaje máquina)

Ejemplo: SAVE "rutina.BIN",49152,65535
Almacena el contenido completo de la memoria

Observación: El sector de la memoria debe ser cargado con el comando RUN"nombre".

Esta instrucción tiene especial interés para grabar programas escritos en lenguaj máquina. Pero esto está reservado para los programadores experimentados, que no están satisfechos con la velocidad del BASIC.

Además puede utilizarse esta instrucción para almacenar gráficos creados. Para ello se transfiere todo el área de la memoria de pantalla (de 40152 a 65535) a un diskette. El almacenamiento de la pantalla gráfica adquiere especial interés, cuando se trata de grabar en diskette una representación matemática complicada, cuya ejecución requiere más tiempo que la grabación. Esto sucede especialmente con funciones tridimensionales.

ENCADENAMIENTO DE PROGRAMAS

Con el paso del tiempo, todos los programadores van elaborando subprogramas artificiosos, que pueden ser aplicados en muchos programas. Una amplia biblioteca de subprogramas reduce considerablemente el trabajo al confeccionar nuevos programas. ¿Pero cómo se encadenan los subprogramas con un programa principal? El amplio vocabulario del CPC6128 también ofrece una solución óptima a este problema.

Problema: Encadenar programas

Instrucción: MERGE "nombre"

Parámetros: nombre - nombre del programa

Ejemplo: MERGE "UPRO1"
Carga el programa "UPRO1.BAS", añadiéndolo al programa ya existente en la memoria.

Observación: Si coinciden los números de línea, se borrarán las líneas del programa residente en la memoria

El siguiente cuadro nos muestra claramente la función de este comando:

Progr. en memoria	Progr. en diskette	Resultado
10 REM P1	5 REM P2	5 REM P2
20 REM P1	15 REM P2	10 REM P1
		15 REM P2
		20 REM P1
10 REM P1	5 REM P2	5 REM P2
20 REM P1	10 REM P2	10 REM P2
		20 REM P1

Aquí se ve claramente, que las líneas de ambos programas se ordenan de tal modo, que el programa resultante corresponde a las prescripciones de la numeración de líneas. Sin embargo, cuando coinciden dos números de línea, se borra la línea del programa que se encuentra previamente en la memoria. Esto es lógico, porque no puede haber números de línea repetidos en un programa.

En el caso de que los números de línea de los dos programa a encadenar no correspondan a las normas de encadenamiento, es necesario adaptarlas previamente. Para ello puede utilizarse p.ej el comando RENUM, explicado anteriormente.

Finalizaremos mostrando un ejemplo de aplicación del comando MERGE. En primer lugar, entre el programa de carga posterior:

```
10 REM programa 2
25 REM programa 2
30 REM programa 2
45 REM programa 2
```

Este es el programa que será cargado más adelante, por lo cual vamos a almacenarlo con la instrucción

SAVE "PROG2"

en un diskette. Después borramos la memoria (NEW) y entramos el programa, al cual queremos añadir el anterior:

```
10 REM programa 1
20 REM programa 1
30 REM programa 1
40 REM programa 1
50 REM programa 1
```

Ahora vamos a encadenar ambos programas, escribiendo la instrucción

MERGE "PROG2"

El ordenador se comporta ahora de modo habitual al cargar un programa. Pero el posterior comando 'LIST' le mostrará lo que ha sucedido realmente:

```
10 REM programa 2
20 REM programa 1
25 REM programa 2
30 REM programa 2
40 REM programa 1
45 REM programa 2
50 REM programa 1
```

Las dos líneas que coinciden -10 y 30- han sido sustituidas por las líneas del segundo programa cargado. Todas las demás han sido ordenadas correctamente. El comando MERGE le será de gran ayuda en la confección de nuevos programas que utilicen subprogramas viejos.

BORRAR FICHEROS

Es aconsejable, que los programas, que han perdido interés para nosotros, no continúen ocupando una parte preciosa de la memoria del diskette. El DOS del CPC, el llamado AMSDOS, dispone de un comando para borrar ficheros de un diskette. Todos los comandos del AMSDOS empiezan por el carácter " | ", que se obtiene al pulsar simultáneamente las teclas SHIFT y ' @ '. Pero veamos ahora el comando:

Problema: **Borrar ficheros**

Instrucción: |ERA,"nombre"

Parámetros: nombre - nombre del fichero

Ejemplo: |ERA,"secret.p"
borra el programa "secret.p"

Pruebe utilizar este comando y borre el programa MERGE del ejemplo anterior:

```
|ERA,"prog2.bas"
```

Tenga en cuenta, que siempre debe indicarse el suplemento .BAS.

REDENOMINAR UN FICHERO

Si en algún momento tiene el deseo de cambiar el nombre de un fichero, utilice la siguiente instrucción del AMSDOS:

Problema: Redenominar un fichero

Instrucción: |REN,"nombrenuevo","nombreantiguo"

Parámetros:

nombrenuevo - nuevo nombre de fichero
nombreantiguo - antiguo nombre de fichero

Ejemplo: |REN,"DEMO.BAS","DEMO01.BAS"
cambia el nombre del programa DEMO1.BAS por DEMO.BAS"

Aquí también es necesario indicar el suplemento del nombre.

Naturalmente el AMSDOS y, en especial, el sistema operativo CP/M, disponen de más comandos que los descritos específicamente en este capítulo. Pero no queremos liar a nuestros lectores, por lo cual este libro sólo se limitará a los comandos más importantes para la gestión de programas en diskettes.

CAPITULO 9:

MAS COMANDOS

Cuando haya llegado felizmente a este capítulo, seguramente estará impaciente por conocer más comandos del extenso BASIC del CPC. No queremos negárselo, y le presentaremos unos comandos, cuya complejidad no es excesiva. Tal como venimos haciendo hasta ahora, cada comando viene acompañado de explicaciones exhaustivas, así como de programas ejemplo.

CONSULTA DEL JOYSTICK

Si se dedica un momento a observar la parte trasera de su CPC, verá a la derecha un conector, que es idéntico en prácticamente todos los ordenadores domésticos. Aquí se conectan los llamados Joysticks, que sirven principalmente para controlar los juegos.

La lectura del Joystick del CPC es relativamente sencilla. Veamos ahora el comando correspondiente:

Problema: Consulta del Joystick

Función: JOY (n)

Parámetros: n - número del Joystick (0 ó 1)

Ejemplo: PRINT JOY(0)
Muestra el código del accionamiento del Joystick en la pantalla.

Observación: Los códigos son los siguientes:

```
5     1     9
                  disparador 1 = 32
4           8
                  disparador 2 = 16
6     2     10
```

Habrá observado, que hay dos Joysticks a consultar, a pesar de que sólo existe un conector. Sólo es posible conectar dos Joysticks, si se trata de Joysticks originales de Amstrad o de otras marcas, que estén destinadas al CPC. Pero si usted no necesita más que uno, puede utilizar cualquier Joystick del comercio (p.ej. de ATARI, COMMODORE64), el cual será llamado por **'JOY(0)'**.

Vamos a escribir ahora un programa, que permite visualizar en la pantalla el accionamiento del Joystick.

```
10 PRINT JOY(0)
20 GOTO 10
```

Al entrar y ejecutar el programa, podrá observar los códigos de cada accionamiento del Joystick. El correspondiente programa podrá responder a estos valores.

Con ayuda de la función **'JOY (n)'** podemos escribir un impresionante programa para dibujar en la pantalla. El disparador será utilizado para borrar la pantalla.

```
10 MODE 2
20 CLS
30 x=320:y=200
40 PLOT x,y
50 a=JOY(0)
60 IF a=4 AND x>0   THEN x=x+1:GOTO 40
70 IF a=8 AND x<639 THEN x=x-1:GOTO 40
80 IF a=1 AND y<399 THEN y=y+1:GOTO 40
90 IF a=2 AND Y>0   THEN y=y-1:GOTO 40
100 IF a=16 THEN CLS
110 GOTO 40
```

La estructura de este programa es muy sencilla, de manera que no tendrá ningún problema para entender su funcionamiento y para efectuar pequeñas modificaciones. Una ampliación interesante sería p.ej. el trazado de diagonales. Un consejo: para ello necesita otras cuatro consultas IF, que deben verificar ambas variables ('x' e 'y'), y actualizarlas con un valor válido.

NUMEROS ALEATORIOS

Casi todos los principiantes del BASIC escriben su primeros juegos utilizando números aleatorios. Sin estos números seriá imposible programar juegos de azar. El CPC dispone de una función, que crea números aleatorios comprendidos entre 0.000000001 y 0.999999999:

Problema: **Crear número aleatorios**

Instrucción: **RND(1)**

Ejemplo: **PRINT RND(1)**
Crea un número aleatorio y lo muestra en la pantalla.

Observación: **El número creado está comprendido entre 0.000000001 y 0.999999999**

Es un comando de fácil aplicación. La complicación empieza, cuando el número buscado debe hallarse dentro de un área determinada. En este caso, es necesario ampliarlo. Por ejemplo, si queremos hallar un número comprendido entre 0 y 999, el comando será el siguiente:

```
10 REM ************
20 REM      dado
30 REM ************
40 numero=REND(1)
50 PRINT INT(numero*1000)
```

Por consiguiente, multiplicamos el número aleatorio por 1000, con lo cual estará comprendido entre 0.000001 y 999.999999. Además le "cortamos" todos los decimales con la función INT, ya que no los necesitamos.

Otra dificultad se presenta, si el número buscado no debe empezar por 0, sino por otro número cualquiera. Pero la siguiente fórmula nos permite limitar cualquier área deseada:

Conversión de números aleatorios
INT (RND(1) * (B+1)-A))+A A - primer número válido B - último número válido

Si queremos obtener, por ejemplo, números comprendidos entre 1 y 6, hay que confeccionar la siguiente fórmula:

```
INT(RND(1) * (6-1)+1))+1
```

equivale a

```
INT(RND(1) * (5+1))+1
```

equivale a

```
INT(RND(1) * 6)+1
```

El último programa generará números de lotería primitiva (6 entre 49):

```
10 REM **********************
20 REM   Numeros de loteria
30 REM **********************
40 PRINT "Cuantas líneas ";lineas
50 FOR i1=1 TO lineas
60    FOR i2=1 TO 6
70       PRINT INT(RND(1)*49)+1;" - ";
80    NEXT i2
90 PRINT
100 NEXT i1
```

No le será difícil entender el funcionamiento de este programa. Pero tiene un fallo: en una misma línea pueden aparecer dos o más números iguales. Usted tendrá ahora la ocasión de evitarlo. Un consejo: Almacene todos los números ya extraídos de una línea en un array (tabla), para compararlos después con el número actual. El número extraído sólo será aceptado en el caso, de que no coincida con ninguno de los almacenados en el array. Sin embargo, hay que volver a borrar el array antes de ir a la línea siguiente. ¡Qué se divierta!

LEFT$

A continuación veremos unas funciones para el tratamiento de Strings, que le abrirán las puertas a nuevas posibilidades. Podrá p.ej. hallar partes aisladas de Strings para su posterior procesamiento, o insertar un String en otro. Es verdaderamente aconsejable estudiar estas funciones. Todas ellas empiezan por su descripción:

Problema: **Hallar String parcial izquierdo**

Función: **LEFT$**(string,cant)

Parámetros: string - **variable de cadena o String entre comillas**

cant - **cantidad de caracteres a considerar, contando a partir de la izquierda.**

Ejemplo: **PRINT LEFT$("ABCDEFG",3)**
Muestra los primeros tres caracteres del String (ABC) en la pantalla.

Observación: **Si la cantidad de caracteres es mayor que el String, se considerará el String completo.**

Por lo tanto, esta interesante función es capaz de "cortar" Strings, empezando por la izquierda. El siguiente programa le explicará más de cien palabras:

```
10 INPUT "Escriba su nombre:";nombre$
20 cant=1
30 parte$=left$(nombre$,cant)
40 PRINT parte$
50 IF parte$=nombre$ THEN 80
60 cant=cant+1
70 GOTO 30
80 PRINT "fin"
```

Puesto que este comando no presenta ninguna dificultad de manejo, ya pasaremos al siguiente:

RIGHT$

Problema:	**Hallar String parcial derecho**	
Función:	**RIGHT$**(string,cant)	
Parámetros:	string -	**variable de cadena o String entre comillas**
	cant -	**cantidad de caracteres a considerar, contando a partir de la derecha.**
Ejemplo:	**PRINT RIGHT$("ABCDEFG",3)** **Muestra los últimos tres caracteres del String (EFG) en la pantalla.**	
Observación:	**Si la cantidad de caracteres supera la longitud del String, se considerará el String completo.**	

Esta función es parecida a la anterior, y no precisa más explicaciones. Si desea vér con mayor claridad la diferencia entre ambas funciones, cambie el programa de la página anterior, sustituyendo la función '**LEFT$**' por '**RIGHT$**'.

Ahora ya hemos hallado la parte izquierda y derecha de un String. Pero existe otra función, que puede incluso separar secciones en el centro de un String:

MID$

Problema: **Averiguar sección de String**

Función: **MID$(string,pos,cant)**

Parámetros: string - **variable de cadena o String entre comillas**

pos - **posición en el String, donde debe empezar la sección**

cant - **cantidad de caracteres a considerar, partiendo de la posición indicada.**

Ejemplo: **PRINT MID$("ABCDEFG",2,4)**
Muestra en la pantalla cuatro caracteres del String, contando desde la posición dos (BCDE).

Observación: **Si la cantidad de caracteres es mayor que el resto del String, se considerará todo el resto.**

Después de recortar los Strings por la izquierda y por la derecha, esta función nos ofrece ahora la posibilidad de hallar secuencias de caracteres del centro del String. Para ello hay que añadir otro parámetro a los dos anteriores, ya que debemos indicar la posición inicial además de la cantidad de caracteres a considerar.

Esta función es utilizada con frecuencia para mostrar cada uno de los caracteres del String. De esta forma podemos visualizar p.ej. un secuencia de caracteres dispuesta verticalmente, tal como muestra el siguiente programa:

```
10 a$="HORIZONTAL"
20 FOR p=1 TO 10: REM Longitud del String
30    PRINT MID$(a$,p,1)
40 NEXT p
```

La posición inicial va desplazándose hacia la derecha con un bucle. La longitud de la sección del String es constante e igual a 1, ya que queremos emitir un solo carácter. ¿Pero qué sucede, si la longitud del String no es conocida? También se habrá resuelto este problema, cuando conozca la siguiente función:

HALLAR LA LONGITUD DE UN STRING

Problema: **Hallar la longitud de un String**

Función: string - **variable de cadena o String entre comillas**

Ejemplo: **A$="ABCDEFG":PRINT LEN(A$)**
Muestra en la pantalla la longitud del String 'A$' (7).

Observación: **Si el String es una cadena vacía, la función da el valor 0.**

Si ahora aplicamos esta función al programa de la página anterior, se emitirá un String cualquiera verticalmente:

```
10 INPUT "String:";a$
20 FOR p=1 TO LEN(a$)
30      PRINT MID$(a$,p,1)
40 NEXT p
```

Esta solución es más elegante que la anterior, ya que no es necesario conocer previamente la longitud del String.

Después de conocer ahora las funciones más importantes de Strings, le presentamos un pequeño programa, que le impresionará:

```
10 REM *********************
20 REM   Escritura continua
30 REM *********************
40 CLS
50 INPUT "Texto:            ";texto$
60 INPUT "Longitud:         ";longitud
70 INPUT "Velocidad (1-7):";velocidad
80 l=LEN(texto$)
90 texto$=STRING$(l,"-")+texto$+String$(l,"-")
100 FOR i=1 TO l
110    LOCATE 12,20
120    PRINT MID$(texto$,i,longitud)
130    FOR wait=1 TO velocidad*50:NREXT wait
140 NEXT i
150 GOTO 100
```

La variable **'texto$'** lee el texto, que debe ser emitido en escritura continua. Después se retiene en **'longitud'** la longitud de la sección de escritura continua. Una vez almacenado un factor de velocidad en **'velocidad'**, la variable **'l'** recibe la longitud del texto. Para separar el texto por caracteres al principio y al final, hemos añadido la línea 90. Aquí se incluye la función **'STRING'** -utilizada en la gestión de direcciones- para crear cadenas de un carácter. El bucle posterior se va ejecutando hasta llegar al final del texto entrado, emitiendo el String parcial, que resulta de la posición y de la longitud indicada. El bucle de espera de la línea 130 se procesa entre 50 y 350, según el valor del factor de velocidad. Este bucle será llamado continuamente, hasta que se interrumpa el programa con la tecla **'ESC'**.

INSTR

Con frecuencia es necesario hallar un contenido determinado dentro de un String, especialmente cuando se trata de la gestión de datos. En primer lugar le decribimos la función, que permite realizar tal tarea.

Problema: **Buscar un String dentro de otro**

Función: INSTR(pos,string1,string2)

Parámetros:

pos - **posición, a partir del cual empieza la búsqueda**

string1 - **String, en el cual se busca**

string2 - **String buscado**

Ejemplo: **PRINT INSTR (3,"ABCDEFGHIJ","FGH")**

Inspecciona el primer String indicado, a partir de la posición 2, buscando el último String indicado, y muestra su posición (6) en la pantalla.

Observación: **Si no se encuentra el String buscado, se emitirá el valor 0. La indicación de la posición puede omitirse, si se trata de inspeccionar el String completo.**

Esta función confortable facilita la búsqueda de ciertos datos en un fichero. Hasta este momento sólo pudimos comprobar Strings en posiciones definidas con exactitud, pero el comando '**INSTR**' nos permite inspeccionar el String completo para hallar una secuencia de caracteres. Vamos a demostrarlo con ayuda de un programa:

```
5 CLS
10 INPUT "TEXTO:                    ";texto$
20 INPUT "EXPRESION A BUSCAR: ";busca$
30 p=INSTR(texto$,busca$)
40 IF p=0 THEN 70
50 PRINT "La expresion a buscar se halla en la  posicion";p;"
del texto"
60 GOTO 80
70 PRINT "Expresion a buscar no hallada"
80 PRINT "Busqueda finalizada"
```

Aquí usted entra primero un texto, seguido de una expresión a buscar. La función INSTR de la línea 30 almacena el resultado de la búsqueda en **'pos'**. En este caso no hemos indicado posición inicial, porque se trata de inspeccionar todo el String **'texto$'**. Unicamente se ha encontrado la expresión buscada, si el resultado de la búsqueda, es decir, si la variable **'pos'** es diferente de cero. En este caso, se emitirá la posición, donde la búsqueda ha tenido éxito.

Para finalizar, veamos ahora una versión ampliada del programa, que permite sustituir la expresión hallada por un concepto nuevo:

```
5 CLS
10 INPUT "TREXTO         ";texto$
20 INPUT "buscar:        ";busca$
30 INPUT "sustituir por:";sus$
40 p=INSTR (texto$,busca$)
50 IF p=0 THEN 90
60 TEXTO$=LEFT$(texto$,p-1)+sus$+MID$(texto$,p+LEN(busca$),
LEN(texto$)-len(busca$)-(p-1))
70 PRINT "Texto nuevo:  ";texto$
80 GOTO 100
90 PRINT "Expresion a buscar no hallada!"
100 PRINT "Busqueda finalizada"
```

La línea 60 puede confundirle un poco, ya que contiene gran número de funciones de cadena. Pero esto es necesario, ya que el nuevo String se compone se las partes siguientes:

1. La parte que precede a la posición del registro hallado (p-1)

2. La nueva expresión

3. El resto, contando a partir de la posición siguiente a la expresión buscada (p+LEN(busca$)) hasta el final del texto (LEN(texto$)-len(busca$)-(p-1))

Podrá encontrar rutinas de este tipo -oara buscar y sustituir- en prácticamente todos los procesadores de texto. El CPC permite realizarla del modo más sencil]o.

APENDICE 1

PALABRAS RESERVADAS

Los nombres de las variables pueden elegirse libremente, pero respetando algunas limitaciones: no se admiten los términos que forman parte del vocabulario BASIC. Si no obedecemos a esta limitación, el CPC nos recuerda con un "Syntax error", lo que no deberíamos olvidar nunca. Si aparece este mensaje en una línea BASIC, sin que usted pueda encontrarle explicación alguna, analice en primer lugar los nombres de las variables. Este error suele tener su causa en la utilización de algún nombre de variable contenido en la lista siguiente:

ABS
AFTER
AND
ASC
ATN
AUTO

BIN$
BORDER

CALL
CAT
CHAIN
CHR$
CINT
CLEAR
CLG
CLOSEIN
CLOSEOUT
CLS
CONT
COS

DATA
DEF
DEFINT
DEFREAL
DEFSTR
DEG
DELETE
DEG
DELETE
DI
DIM
DRAW
DRAWR

EDIT
EI
ELSE
END
ENT
ENV
EOF
ERASE
ERROR
EVERY
EXP

FIX
FN
FOR
FRE

GOSUB
GOTO

HEX$
HIMEM

IF
INK
INKEY
INKEY$
INP
INPUT
INSTR

CREAL

JOY

KEY

LEFT$
LEN
LET
LINE
LIST
LOAD
LOCATE
LOG
LOG10
LOWER$

MAX
MEMORY
MERGE
MID$
MIN
MOD
MODE
MOVE
MOVER

NEXT
NEW
NOT

ON
ON BREAK
ON ERROR GOTO
ON SQ
OPENIN
OPENOUT
OR
ORIGIN

ERL
ERR
OUT
PAPER
PEEK
PEN
PI
PLOT
PLOTR
POKE
POS
PRINT

RAD
RANDOMIZE
READ
RELEASE
REM
REMAIN
RENUM
RESTORE
RESUME
RETURN
RIGHT$
RND
ROUND
RUN

SAVE
SGN
SIN
SOUND
SPACE$
SPC
SPEED
SQ
SQR
STEP
STOP

INT

STR$
STRING$
SWAP
SYMBOL

TAB
TAG
TAGOFF
TAN
TEST
TESTR
THEN
TIME
TO
TROFF
TRON

UNT
UPPER$
USING

VAL
VPOS

WAIT
WEND
WHILE
WIDTH
WINDOW
WRITE

XOR
XPOS

YPOS

ZONE

APENDICE 2

MENSAJES DE ERROR

Por muy extraño que pueda parecer: los mensajes de error son tan importantes para el ordenador como los propios comandos. A mayor diversidad de mensajes de error disponibles en un ordenador, mayor facilidad tiene el usuario para localizar su error. Imagínese, que su ordenador contestase con "Syntax error" a todos los errores que usted cometa. En este caso seguramente tendría muchas dificultades para detectar el tipo de error cometido. Sin embargo, el CPC puede emitir gran número de mensajes de error, que describiremos a continuación. De esta forma podrá consultar la lista siempre que se tropiece con un error desconocido.

El número, que se indica delante de cada mensaje, es necesario para interceptar el error dentro de un programa (ON ERROR GOTO). Pero este comando no ha sido explicado en el presente libro, de modo que aprenderá más adelante a utilizar los números.

1 Unexpected NEXT

El programa contiene un NEXT, que no va precedido de un bucle FOR, o bien la variable de bucle del NEXT no coincide con la del comando FOR.

2 Syntax error

Este mensaje de error no será nuevo para usted. Se da en aquellos casos, en que el CPC no pueda entender la línea que usted ha entrado.

3 Unexpected RETURN

La ejecución del programa ha finalizado en un RETURN, ya que no ha sido precedido de un comando GOSUB.

4 DATA exhausted

El comando READ debe leer más datos, que los indicados en la línea DATA.

5 Improper argument

Es un mensaje de error general, que aparece siempre que el valor de una función o de un parámetro no sea válido.

6 Overflow

El resultado de una operación supera el máximo número admisible en la memoria, o la asignación para una variable entera no puede ser codificada como número entero de 16 Bits.

7 Memory full

El programa o las variables han alcanzado una extensión, que supera la memoria disponible. El comando MEMORY provoca este mensaje, cuando se intenta colocar el principio del BASIC en un área no admitida.

8 Line does not exist

Un número de línea indicado no existe en el programa.

9 Subscript out of range

El índice de un array es demasiado pequeño, o es superior al reservado con DIM.

10 Array already dimensioned

Un array de la instrucción DIM ya ha sido dimensionado.

11 Division by zero

Se ha efectuado una división entre cero, no permitida matemáticamente.

12 Invalid direct command

Se ha entrado en modo directo una instrucción, que sólo puede aparecer en programas (p.ej. INPUT).

13 Type mismatch

Se ha indicado un valor numérico, allí donde se eperaba un String, o viceversa.

14 String space full

No hay más memoria disponible para crear Strings.

15 String too long

Se ha superado el límite de 255 caracteres de un String p.ej. por una suma de Strings.

16 String expression too long

Para hallar expresiones de cadena extensas, se almacenan Strings en una memoria temporal. Al superar el límite de esta memoria, aparece el mensaje de error.

17 Cannot CONTinue

No se puede reanudar la ejecución del programa con el comando CONT.

18 Unknown user function

Una función (FN) no ha sido definida con DEF FN.

19 RESUME missing

El programa ha sido interrumpido en una rutina ON ERROR GOTO.

20 Unexpected RESUME

Un RESUME no puede ser efectuado, ya que no se ha accedido a una rutina ON ERROR GOTO.

21 Direct command found

Se ha hallado una línea sin número correspondiente, al cargar un programa de cassette.

22 Operand missing

No se ha indicado un parámetro indispensable.

23 Line too long

Una línea BASIC supera la extensión máxima (255 caracteres).

24 EOF met

Se ha intentado leer un fichero más allá de su final.

25 File type error

El acceso de lectura a un programa o un fichero es erróneo. Así no se puede leer p.ej. un fichero con el comando LOAD.

26 NEXT missing

Se ha programado un bucle FOR sin el correspondiente NEXT.

27 File already open

Se ha intentado abrir un fichero, que todavía no ha sido cerrado.

28 Unknown command

Se ha entrado un comando, que no existe en el vocabulario.

29 WEND missing

Se ha utilizado un bucle WHILE, sin el correspondiente WEND.

30 Unexpected WEND

El programa tropieza con un comando WEND, que no va precedido del correspondiente bucle WHILE.

APENDICE 3:

Indice alfabético

COMMODORE



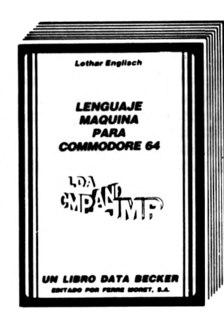
Ofrece un campo fascinante y amplio de problemáticas científicas. Para esto el libro contiene muchos listados interesantes: Análisis de Fornier y síntesis, análisis de redes, exactitud de cálculo, formateado de números, cálculo del valor PH, sistemas de ecuaciones diferenciales, modelo ladrón presa, cálculo de probabilidad, medición de tiempo, integración, etc.
64 en el campo de la Técnica y la Ciencia. 361 págs. P.V.P. 2.800,- ptas.



La obra Standard del floppy 1541, todo sobre la programación en disquettes desde los principiantes a los profesionales, además de las informaciones fundamentales para el DOS, los comandos de sistema y mensajes de error, hay varios capítulos para la administración práctica de ficheros con el FLOPPY, amplio y documentado Listado del Dos. Además un filón de los más diversos programas y rutinas auxiliares, que hacen del libro una lectura obligada para los usuarios del Floppy.
Todo sobre el Floppy 1541. Precio venta 3.200 ptas.



Un excelente libro, que le mostrará todas las posibilidades que le ofrece su grabadora de casettes. Describe detalladamente, y de forma comprensible, todo sobre el Datassette y la grabación en cassette. Con verdaderos programas fuera de serie: Autostart, Catálogo (¡busca y carga automáticamente!), backup de y a disco, SAVE de áreas de memoria, y lo más sorprendente: un nuevo sistema operativo de cassette con el 10-20 veces más rápido Fast Tape. Además otras indicaciones y programas de utilidad (ajuste de cabezales, altavoz de control).
El Manual del Cassette. 190 pág. P.V.P. 1.600,- ptas.



¡Por fin una introducción al código máquina fácilmente comprensible! Estructura y funcionamiento del procesador 6510, introducción y ejecución de programas en lenguaje máquina, manejo del ensamblador, y un simulador de paso a paso escrito en BASIC.
Lenguaje máquina para Commodore 64. 1984, 201 pág. P.V.P. 2.200,- ptas.



CONSEJOS Y TRUCOS, con más de 70.000 ejemplares vendidos en Alemania, es uno de los libros más vendidos de DATA BECKER. Es una colección muy interesante de ideas para la programación del Commodore 64, de POKEs y útiles rutinas e interesantes programas. Todos los programas en lenguaje máquina con programas cargadores en Basic.
64 Consejos y Trucos. 1984, 364 pág. P.V.P. 2.800,- ptas.



Este libro, contiene muchos interesantes programas de aprendizaje para solucionar problemas, descritos detalladamente y de manera fácilmente comprensible. Temas: progresiones geométricas, palanca mecánica, crecimiento exponencial, verbos irregulares, ecuaciones de segundo grado, movimientos de péndulo, formación de moléculas, aprendizaje de vocablos, cálculo de interés y su capitalización.
Manual escolar para su Commodore 64. 389 págs. P.V.P. 2.800,- ptas.



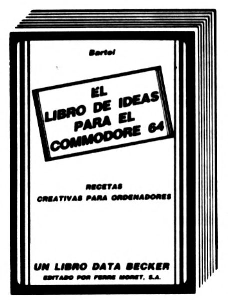
En el libro de los robots se muestran las asombrosas posibilidades que ofrece el CBM 64, para el control y la programación, presentadas con numerosas ilustraciones e intuitivos ejemplos. El punto principal: Cómo puede construirse uno mismo un robot sin grandes gastos. Además, un resumen del desarrollo histórico del robot y una amplia introducción a los fundamentos cibernéticos. Gobierno del motor, el modelo de simulación, interruptor de pantalla, el Port-Usuario cómodo del modelo de simulación, Sensor de infrarrojos, concepto básico de un robot, realimentación unidad cibernética, Brazo prensor, Oír y ver.
Robótica para su Commodore 64. 340 págs. P.V.P. 2.800 ptas.



Saberse apañar uno mismo, ahorra tiempo, molestias y dinero, precisamente problemas como el ajuste del floppy o reparaciones de la platina se pueden arreglar a menudo con medios sencillos. Instrucciones para eliminar la mayoría de perturbaciones, listas de piezas de recambio y una introducción a la mecánica y a la electrónica de la unidad de disco, hay también indicaciones exactas sobre herramientas y material de trabajo. Este libro hay que considerarlo en todos sus aspectos como efectivo y barato.
Mantenimiento y reparación del Floppy 1541. 325 págs. P.V.P. 2.800,- ptas.



Este es el libro que buscaba: un diccionario general de micros que contiene toda la terminología informática de la A a la Z y un diccionario técnico con traducciones de los términos ingleses de más importancia - los DICCIONARIOS DATA BECKER prácticamente son tres libros en uno. La increíble cantidad de información que contienen, no sólo los convierte en enciclopedias altamente competente, sino también en herramientas indispensables para el trabajo. El DICCIONARIO DATA BECKER se edita en versión especial para APPLE II, COMMODORE 64 e IBM PC.
El diccionario para su Commodore 64. 350 pág. P.V.P. 2.800,- ptas.



Casi todo lo que se puede hacer con el Commodore 64, está descrito detalladamente en este libro. Su lectura no es tan sólo tan apasionante como la de una novela, sino que contiene, además de listados de útiles programas, sobre todo muchas, muchas aplicaciones realizables en el C64. En parte hay listados de programas listos para ser tecleados, siempre que ha sido posible condensar «recetas» en una o dos páginas. Si hasta el momento no sabía que hacer con su Commodore 64, ¡después de leer este libro lo sabrá seguro!
El libro de Ideas del Commodore 64. 1984, más de 200 páginas, P.V.P. 1.600,- ptas.



¿Ud. ha logrado iniciarse en código máquina? Entonces el «nuevo English» le enseñará cómo convertirse en un profesional. Naturalmente con muchos programas ejemplo, rutinas completas en código máquina e importantes consejos y trucos para la programación en lenguaje máquina y para el trabajo con el sistema operativo.
Lenguaje máquina para avanzados CBM 64. 1984, 206 pàg. P.V.P. 2.200 ptas.



Este libro ofrece una amplia práctica introducción en el importante tema de la gestión de ficheros y bancos de datos, especialmente para los usuarios del Commodore 64. Con muchas interesantes rutinas y una confortable gestión de ficheros.
Todo sobre bases de datos y gestión de ficheros para Commodore-64. 221 págs. P.V.P. 2.200,- ptas.



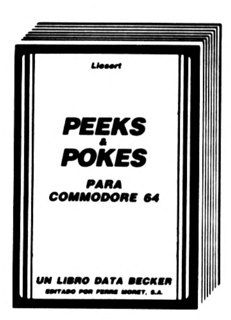
Gráficos para el Commodore 64 es un libro para todos los que quieren hacer algo creativo con su ordenador. El contenido abarca desde los fundamentos de la programación de gráficos hasta el diseño asistido por ordenador (CAD).
Gráficos para el Commodore 64. 295 pags. P.V.P. 2.200,- ptas.



Para los usuarios que posean un VIC-20, C-64 o PC-128 este libro contiene gran cantidad de consejos, trucos, listados de programas, así como información sobre Hardware, tanto si usted dispone de una impresora de margarita o de matriz, como si tiene un Plotter VC-1520, el GRAN LIBRO DE IMPRESORAS constituye una inestimable fuente de información.
Todo sobre Impresoras. 361 págs. P.V.P. 2.800,- ptas.



Con más de 60.000 ejemplares vendidos, ésta es la obra estándar para el COMMODORE 64. Todo sobre la tecnología, el sistema operativo y la programación avanzada del C-64. Con listado completo y exhaustivo de la ROM, circuitos originales documentados y muchos programas. ¡Conozca su C-64 a fondo!
64 Interno. 1984, 352 pág. P.V.P. 3.800,- ptas.



Con importantes comandos PEEK y POKE se pueden hacer también desde el Basic muchas cosas, para las que se necesitarían normalmente complejas rutinas en lenguaje máquina. Con una enorme cantidad de POKEs importantes y su posible aplicación. Para ello se explica perfectamente la estructura del Commodore 64: Sistema operativo, interpretador, página cero, apuntadores y stacks, generador de caracteres, registros de sprites, programación de interfaces, desactivación de interrupt. Además una introducción al lenguaje máquina. Muchos programas ejemplo.
PEEKS y POKES. 177 pág. P.V.P. 1.600,- ptas.

RESPUESTA
COMERCIAL

F.D. Autorización 6975
(B.O. de Correos N.º 80 de 26-7-85)

HOJA PEDIDO
DE LIBRERIA

NO NECESITA
SELLOS

A franquear
en destino

FERRE MORET, S.A.

Apartado N.º 551. F.D.
08080 BARCELONA

RESPUESTA
COMERCIAL

F.D. Autorización 6975
(B.O. de Correos N.º 80 de 26-7-85)

HOJA PEDIDO
DE LIBRERIA

NO NECESITA
SELLOS

A franquear
en destino

FERRE MORET, S.A.

Apartado N.º 551. F.D.
08080 BARCELONA

Puesta al día de datos

EDITORIAL FERRER MORET, S.A. mantiene vivo y amplía el contenido informativo de sus libros y programas, mediante el envío de un servicio de puesta al día, junto con una síntesis noticiosa de la actualidad y perspectivas de la realidad informática española.
Agradecemos cualquier sugerencia o crítica que desee formular y que nos ayude a mejorar las ediciones.
Muchas gracias.

¿Qué añadiría?
....

....

¿Qué suprimiría?
....

....

Observaciones
....

....

Título del libro
....

Nombre
....

Dirección Tfno.
....

Código Postal y Población Provincia
....

UN SERVICIO GRATUITO

Información

FERRE MORET, S.A. cuenta con un amplio fondo de libros y Software y mantiene un servicio de información por correo sobre las novedades que edita.
Agradecemos nos indique los temas que representan para Vd. mayor interes.

Libros	☐ ATARI	☐ MSX	☐ AMSTRAD
	☐ COMMODORE	☐ LENGUAJES	☐ APPLE
	☐ SINCLAIR	☐ IBM	☐ SOFTWARE

Si está interesado en recibir alguno de estos servicios, rellene y envíe la tarjeta correspondiente; no necesita franqueo. Muchas gracias.

DESEO RECIBIR EL LIBRO ....

EL PROGRAMA ....

☐ Adjunto cheque ☐ Contra reembolso
....

Nombre
....

Dirección Tfno.
....

Código Postal y Población Provincia
....

UN SERVICIO GRATUITO

EL CONTENIDO:

El libro CPC 6128 PARA PRINCIPIANTES debería ser el primer libro sobre el CPC 6128. Se trata de una introducción fácilmente comprensible al manejo, aplicación y programación del AMSTRAD CPC 6128, que no exige ningún conocimiento previo.

Extracto del contenido:

— El manejo del teclado
— La primera sentencia
— El primer programa
— Introducción BASIC paso a paso con confección de una completa gestión de direcciones
— Sentencias GRAFIK con programa de ejemplo
— Comandos SOUND con impresionantes programas de demostración (p. ej. órgano en el teclado)
— La unidad de disco y su manejo
— Otras sentencias útiles (p. ej. lectura del joystick, números aleatorios)

ESTE LIBRO HA SIDO ESCRITO POR:

Norbert Szczepanowski, informático y autor de Betsellers de DATA BECKER. Tiene varios años de experiencia en la programación de numerosos ordenadores.

ISBN 84-86437-54-7

Norbert Szczepanowski / CPC 6128 para principiantes

AMSTRAD


AMSTRAD
CPC
OCR
MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA
ACME
https://acpc.me/